LISFLOOD-FP 8.2: GPU-accelerated multiwavelet discontinuous Galerkin solver with dynamic resolution adaptivity for rapid, multiscale flood simulation

Alovya Ahmed Chowdhury[1], Georges Kesserwani[1]

Department of Civil and Structural Engineering, University of Sheffield, Sheffield, S10 2TN, United Kingdom

Correspondence: Alovya Chowdhury (alovya.chowdhury@gmail.com) and Georges Kesserwani (g.kesserwani@sheffield.ac.uk)

Abstract. The second-order discontinuous Galerkin (DG2) solver of the two-dimensional (2D) shallow water equations in the raster-based LISFLOOD-FP 8.0 hydrodynamic modelling framework is mostly suited for predicting small-scale transients that emerge in rapid, multiscale floods caused by impact events like tsunamis. However, this DG2 solver can only be used for simulations on a uniform grid where it may yield inefficient runtimes even when using its graphics processing unit (GPU) parallelised version (GPU-DG2). To boost efficiency, the new LISFLOOD-FP 8.2 version integrates GPU parallelised dynamic (in time) grid resolution adaptivity of multiwavelets (MW) with the DG2 solver (GPU-MWDG2). The GPU-MWDG2 solver performs dyadic grid refinement, starting from a single grid cell, with a maximum refinement level, L, based on the resolution of the Digital Elevation Model (DEM). Furthermore, the dynamic GPU-MWDG2 adaptivity is driven by one error threshold, ε, against normalised details of all prognostic variables. Its accuracy and efficiency, as well as the practical validity of recommended ε choices between 10^{-4} and 10^{-3}, are assessed for four laboratory/field-scale benchmarks of tsunami-induced flooding with different impact event complexities (i.e. single- vs. multi-peaked) and L values. Rigorous accuracy and efficiency metrics consistently show that GPU-MWDG2 simulations with $\varepsilon = 10^{-3}$ preserve the predictions of the GPU-DG2 simulation on the uniform DEM grid, whereas $\varepsilon = 10^{-4}$ may slightly improve velocity-related predictions. Efficiency-wise, GPU-MWDG2 yields considerable speedups from $L \geq 10$—due to its scalability on the GPU with increasing L—which can be around 2.0-to-4.5-fold. Generally, the bigger the $L \geq 10$, the lower the event complexity over the simulated duration, and the closer the ε to 10^{-3}, the larger the GPU-MWDG2 speedups over GPU-DG2. The LISFLOOD-FP 8.2 code is open source, under the GPL v3.0 licence, as well as the simulated benchmarks' set-up files and datasets, with a video tutorial and further documentation on https://www.seamlesswave.com/Adaptive (last accessed: 6 July 2025).

1 Introduction

LISFLOOD-FP is a raster-based hydrodynamic modelling framework that has been used to support various geoscientific modelling applications (Hajihassanpour et al., 2019; Hunter et al., 2005; Nandi & Reddy, 2022; Zeng et al., 2022; Ziliani et al., 2020). LISFLOOD-FP has a suite of numerical solvers of the two-dimensional shallow water equations, for which the prognostic variables are the time-variant water depth h (m) and unit-width discharges hu and hv (m^2 s^{-1}) and time-invariant topography z (m), all represented as raster-formatted data on a uniform grid at a Digital Elevation Model (DEM) resolution. It includes a diffusive wave solver (Hunter et al., 2005), a local inertial solver (Bates et al., 2010), a first-order finite volume solver, and a second-order discontinuous Galerkin (DG2) solver (Shaw et al., 2021), all already supported with robustness treatments (i.e. for topographic and friction discretisation with wetting and drying), ensuring numerical mass conservation to machine precision. The DG2 solver is the most complex numerically, requiring three times more modelled data (degrees of freedom) per prognostic variable and at least twelve times more computations per cell compared to any of the other solvers in LISFLOOD-FP. This complexity arises from its locally conservative formulation that pays off with more inherent mimetic properties (Ayog et al., 2021; Kesserwani et al., 2018), i.e. numerical momentum conservation and reduced (spurious) error dissipation, leading to more accurate velocity fields and long-duration flood simulations (Ayog et al., 2021; Kesserwani & Sharifian, 2023; Sun et al., 2023). Thus, even when parallelised on a graphics processing unit (GPU), the GPU parallelised DG2 solver (GPU-DG2) may still have prohibitively long runtimes, e.g. when applied to run simulations at DEM resolutions close to 1 m and/or with DEM grid sizes beyond 1 km (Kesserwani & Sharifian, 2023; Shaw et al., 2021).

DG2 simulations were shown to accurately reproduce slow to gradual fluvial/pluvial flooding flows at unusually coarse DEM resolutions (Ayog et al., 2021; Kesserwani, 2013; Kesserwani & Wang, 2014), but they primarily excel at capturing the small-scale rapid flow transients that occur over a wide range of spatial and temporal scales (Kesserwani & Sharifian, 2023; Sharifian et al., 2018; Sun et al., 2023). Such transients are typical of flooding flows driven by impact event(s) like tsunami(s) and including zones of flow recirculation past (un)submerged island(s). Hence, DG2 simulations are likely suited for obtaining accurate modelling of rapid, multiscale flooding, such as for tsunami-induced inundations. Within this scope, dynamic (in time) mesh adaptivity has often been deployed with finite volume based tsunami simulators to reduce simulation runtimes (Lee, 2016; Popinet, 2012). This paper reports the integration of dynamic grid resolution adaptivity into the GPU-DG2 solver in a new LISFLOOD-FP 8.2 release to reduce the runtimes of rapid, multiscale flow simulations, which are here exemplified by tsunami-induced flooding events.

Unlike static grid resolution adaptivity—integrated into LISFLOOD-FP 8.1 with the reduced-complexity finite volume solver (Sharifian et al., 2023)—this LISFLOOD-FP 8.2 version performs dynamic grid resolution adaptivity with the DG2 solver every simulation timestep to achieve as much local grid resolution coarsening as possible. Namely, grid coarsening is applied to the grid cells covering either regions of smooth flow or DEM features, thereby reducing the number of computational cells as much as allowed by the complexity of the DEM and flow solution. The LISFLOOD-FP 8.2 version is unique in providing a single, mathematically sound hydrodynamic modelling framework that implements dynamic grid

resolution adaptivity entirely on the GPU for achieving raster-grid DG2 simulations that can preserve a similar level of predictive accuracy and robustness as alternative GPU-DG2 simulations on a uniform grid. The framework is formulated in Kesserwani & Sharifian (2020), having been informed by Caviedes-Voullième & Kesserwani (2015), Gerhard et al. (2015), Kesserwani et al. (2015) and Caviedes-Voullième et al. (2020), so as to preserve all the mimetic properties inherent in the reference DG2 hydrodynamic solver (Kesserwani et al., 2018; Kesserwani & Sharifian, 2023). Here, it has been computationally optimised and integrated into LISFLOOD-FP 8.2, in which dynamic grid resolution adaptivity of multiwavelets (MW) is combined with the DG2 solver formulation. This combination is hereafter referred to as dynamic GPU-MWDG2 adaptivity or the GPU-MWDG2 solver.

Existing hydrodynamic modelling frameworks that have integrated dynamic grid resolution adaptivity with GPU parallelisation were mostly based on finite volume solvers (Berger et al., 2011; de la Asunción & Castro, 2017; Ferreira & Bader, 2017; Kevlahan & Lemarié, 2022; LeVeque et al., 2011; Liang et al., 2015; J. Park et al., 2019; Popinet, 2011, 2012; Popinet & Rickard, 2007). Comparatively, fewer are based on DG solvers that are mostly developed for tsunami inundation simulation on triangular or curvilinear meshes, with sparse focusses: either on achieving parallelisation on central processing units (CPU) with extrinsic forms of dynamic resolution adaptivity, or on parallelising non-adaptive DG solvers on the GPU; while, in any of the focusses, addressing robustness treatments (Blaise et al., 2013; Blaise & St-Cyr, 2012; Bonev et al., 2018; Castro et al., 2016; Hajihassanpour et al., 2019; Rannabauer et al., 2018). To mention just a few, Blaise & St-Cyr (2012) and Blaise et al. (2013) integrated CPU parallelisation into dynamic adaptivity for curvilinear meshes, calling for better forms of adaptivity with improved robustness treatments to achieve reliable DG based tsunami inundation simulations. Rannabauer et al. (2018) addressed wetting and drying treatments with a DG based solver of tsunami inundation that integrated CPU parallelised dynamic adaptivity on triangular meshes; further, the authors identified the benefit of their DG based solver compared to the finite volume solver counterpart. To track tsunami propagation on the sphere, Bonev et al. (2018) and Hajihassanpour et al. (2019) devised DG based solvers with dynamic adaptivity for curvilinear meshes, highlighting the need to further exploit GPU parallelisation to achieve realistic runtimes. For tsunami inundation simulations, Castro et al. (2016) found that non-adaptive DG simulations on triangular meshes to be 23-fold faster when parallelised on the GPU compared to parallelised simulations on the CPU with 24 threads.

Yet, to the best of the authors' knowledge, there is no existing DG based hydrodynamic modelling framework that combines raster grid-based dynamic resolution adaptivity with GPU parallelisation within a mathematically sound framework, i.e. one that preserves the mimetic properties of the reference robust DG2 solver on the uniform grid (Kesserwani et al., 2019; Kesserwani & Sharifian, 2020, 2023) like the dynamic GPU-MWDG2 adaptivity integrated into LISFLOOD-FP 8.2. To elaborate, the MWDG2 solver automates local grid resolution coarsening on an adaptive grid using the multiresolution analysis (MRA) of MW applied to scaled DG2 modelled data (for all the prognostic variables) that exist in a hierarchy of grids. This hierarchy consists of dyadically coarser grids relative to the finest grid in the hierarchy (Kesserwani et al., 2019; Kesserwani & Sharifian, 2020; Sharifian et al., 2019), whereby the finest resolution grid is associated with a maximum refinement level, L, and consists of $(2^L \times 2^L)$ cells—with L selected to match the resolution of a

raster-formatted DEM file. Meanwhile, the coarsest resolution grid consists of $2^0 \times 2^0 = 1$, i.e. a single cell. On the hierarchy of grids, the scaled DG2 modelled data for each prognostic variable are compressed into higher-resolution MW coefficients, or *details*, which are added to the coarsest resolution data. The details of all the prognostic variables are normalised and packed in a dataset of normalised details, which are then compared to an error threshold $0 < \varepsilon < 1$ for retaining the significant details. The retained significant details are added to the coarsest resolution DG2 modelled data, leading to a multiscale representation of DG2 modelled data on a non-uniform grid. As the scaling, analysis, and reconstruction of DG2 modelled data are inherent to the MRA procedure, the mimetic properties of the reference GPU-DG2 solver on the finest resolution grid ($2^L \times 2^L$) can readily be preserved for $\varepsilon \leq 10^{-3}$, based on studies considering a range for ε between 10^{-6} to 10^{-1} (Kesserwani et al., 2019; Kesserwani & Sharifian, 2020). Another benefit of the MRA procedure is the sole reliance on ε to sensibly control the amount of local grid resolution coarsening for all prognostic variables. Furthermore, it was also shown that, for the same ε, dynamic MWDG2 adaptivity avoids unnecessary refinement compared to its first-order counterpart, and using $\varepsilon \geq 10^{-4}$ leads to simulations that are faster than first-order finite volume solvers on the finest resolution grid (Kesserwani & Sharifian, 2020).

The dynamic MWDG2 adaptivity integrated in LISFLOOD-FP 8.2 optimises the approach proposed in Kesserwani & Sharifian (2020), which was initially designed for realistic, two-dimensional hydrodynamic modelling on a single core CPU and was based on the reference DG2 hydrodynamic solver in Kesserwani et al. (2018). A. A. Chowdhury et al. (2023) devised a computationally efficient GPU implementation of wavelet adaptivity integrated with a first-order finite volume (FV1) solver counterpart. They reported that the speedup of their GPU implementation over the uniform FV1 solver run on the finest $2^L \times 2^L$ resolution grid scaled up with increasing L, becoming considerable for $L \geq 9$. In fact, for $L \geq 9$, with the same global timestep Δt used, there would be enough memory and compute workload to bound the GPU, which the adaptive solver can reduce unlike the uniform solver. Kesserwani & Sharifian (2023) extended the GPU implementation of A. A. Chowdhury et al. (2023) to produce the GPU-MWDG2 solver. They analysed its efficiency for $\varepsilon = 10^{-3}$ in simulating realistic flooding flow scenarios for adaptive grids involving $L \geq 10$. Their findings revealed that the GPU-MWDG2 solver is 3-fold faster than the GPU-DG2 solver on the finest resolution grid for a rapid flood scenario driven by an impact event, where dynamic GPU-MWDG2 adaptivity did not use more than 85% of the number of cells of GPU-DG2. Hence, a dedicated study is needed to analyse the usability and practical merit of GPU-MWDG2 simulations over GPU-DG2 simulations for rapid multiscale flooding scenarios.

Next, in Sect. 2, the GPU-MWDG2 solver in LISFLOOD-FP 8.2 is described with a focus on its use for running GPU-MWDG2 simulations from raster-formatted DEM and initial flow setup files (Sect. 2.1), its associated upper memory limits and scalability (Sect. 2.2), and analysis of efficiency using newly proposed metrics (Sect. 2.3). In Sect. 3, the efficiency and accuracy of the GPU-MWDG2 solver are assessed against the GPU-DG2 solver, with choices of $\varepsilon = 10^{-3}$ and 10^{-4}, for four tsunami-induced flooding benchmarks at both laboratory- and field-scale. Sect. 4 draws conclusions and recommendations as to when the GPU-MWDG2 solver can best be used over the GPU-DG2 solver. The LISFLOOD-FP 8.2 code is open-source, under the GPL v3.0 licence (LISFLOOD-FP developers, 2024), as well as the simulated benchmarks'

set-up files and datasets (A. A. Chowdhury & Kesserwani, 2024), a video tutorial (A. Chowdhury, 2025) and documentation on https://www.seamlesswave.com/Adaptive (last accessed: 7 July 2025).

2 LISFLOOD-FP 8.2

LISFLOOD-FP 8.2 includes the new capability of running simulations over a non-uniform grid using dynamic GPU-MWDG2 adaptivity. The GPU-MWDG2 solver can be used as an alternative to the uniform-grid GPU-DG2 solver (Shaw et al., 2021) to potentially reduce simulation runtimes. Unlike with LISFLOOD-FP 8.1, where the MRA procedure of MW is only applied once at the beginning of the simulation to generate a static non-uniform grid whose grid resolution is locally coarsened as much as permitted by features of the time-invariant DEM features (Sharifian et al., 2023), the GPU-MWDG2 solver deploys the MRA procedure every simulation timestep, denoted by Δt, to also automate grid resolution coarsening based on the features of the time-varying flow solution.

The technical description of the GPU-MWDG2 solver has been reported in previous papers (Kesserwani & Sharifian, 2020, 2023), whose dynamic adaptivity has here been further computationally optimised to improve memory coalescing and occupancy in the CUDA kernels (NVIDIA, 2023). Therefore, the GPU-MWDG2 solver is briefly overviewed in Appendix A, with a focus on its operational workflow with reference to Figure 1. In what follows, the presentation is focussed on describing the features incorporated into LISFLOOD-FP 8.2 for running the GPU-MWDG2 solver (Sect. 2.1), on identifying the upper limits of its GPU memory consumption in relation to the specification of the GPU card (Sect. 2.2), and on proposing metrics for detailed analysis of the efficiency of its dynamic adaptivity from output datasets (Sect. 2.3).

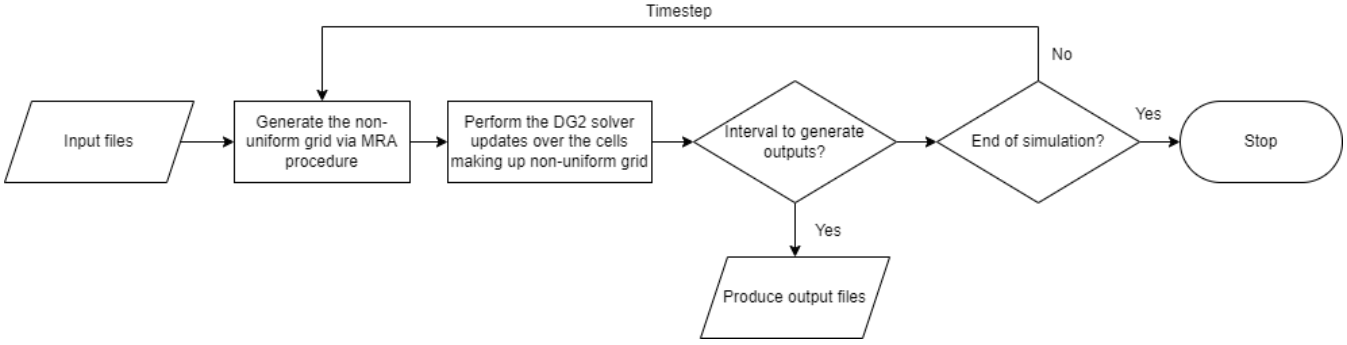

Figure 1: The main operations involved in the GPU-MWDG2 solver (further detailed in Appendix A).

2.1 The GPU-MWDG2 solver

Running a simulation of a test case using any solver in LISFLOOD-FP requires setting up several test case-specific input files[1]. The same is required for the GPU-MWDG2 solver. An important input file is the "parameter" file with the extension

[1] https://www.seamlesswave.com/Merewether1; https://www.seamlesswave.com/Adaptive

`.par`, which is a text file specifying various solver and simulation parameters[2]. In the remainder of this paper, the usability of the GPU-MWDG2 solver will be described for the "Monai valley" test case (explored in Sect. 3.1) without loss of generality. Step-by-step instructions on how to use the GPU-MWDG2 solver to run a simulation of the "Monai valley" test case have been provided in Appendix B.

```
1  cuda
2  mwdg2
3  epsilon       0.001
4  max_ref_lvl   9
5  wall_height   0.5
6  refine_wall
7  ref_thickness 16
8  initial_tstep 1
9  cumulative
10 raster_out
11 fpfric        0.01
12 sim_time      22.5
13 massint       0.1
14 saveint       22.5
15 DEMfile       monai.dem
16 startfile     monai.start
17 bcifile       monai.bci
18 bdyfile       monai.bdy
19 stagefile     monai.stage
```

Figure 2: Listing of parameters in the .par file needed to run a GPU-MWDG2 simulation for the "Monai Valley" test case (Sect. 3.1), with the GPU-MWDG2 specific items highlighted in bold.

The parameters[2] or keywords that should be typed in the `.par` file for running a simulation of the Monai Valley test case are shown in Figure 2, including seven keywords related to running the GPU-MWDG2 solver highlighted in bold. The `cuda` keyword should be typed to access the GPU parallelised models in LISFLOOD-FP, e.g. the GPU-DG2 solver or the GPU-MWDG2 solver. The `mwdg2` keyword should be typed to select the GPU-MWDG2 solver[3]. The `epsilon` keyword should be followed by a numerical value, e.g. 0.001 specifies the error threshold $\varepsilon = 10^{-3}$. The `max_ref_lvl` keyword followed by an integer value specifies the maximum refinement level L, which is specified according to the DEM size and resolution, as explained next.

The GPU-MWDG2 solver starts a simulation on a *square uniform grid* made up of $2^L \times 2^L$ cells, which is the finest-resolution grid accessible to the GPU-MWDG2 solver (Appendix A). In practice however, the DEM may often involve a (rectangular) grid with M rows and N columns, for which the GPU-MWDG2 solver still generates a starting *square uniform grid* with $2^L \times 2^L$ cells. A suitable choice for L is a value such that $2^L \geq max(N, M)$. For example, in the Monai valley test case, $N = 784$ and $M = 486$, leading to $L = 10$. Figure 3 shows the initial non-uniform grid generated by the GPU-MWDG2 solver. Since the GPU-MWDG2 solver starts from a *square uniform grid* inclusive of the DEM dimensions, two areas emerge in the non-uniform grid: the actual test case area containing the flow domain, which includes the DEM data and the initial flow conditions; and, empty areas where no DEM data are available and where no flow should occur.

[2] https://www.seamlesswave.com/Merewether1-1.html; https://github.com/al0vya/gpu-mwdg2
[3] The CPU version of the MWDG2 solver was not integrated on LISFLOOD-FP due to its uncompetitive runtimes.

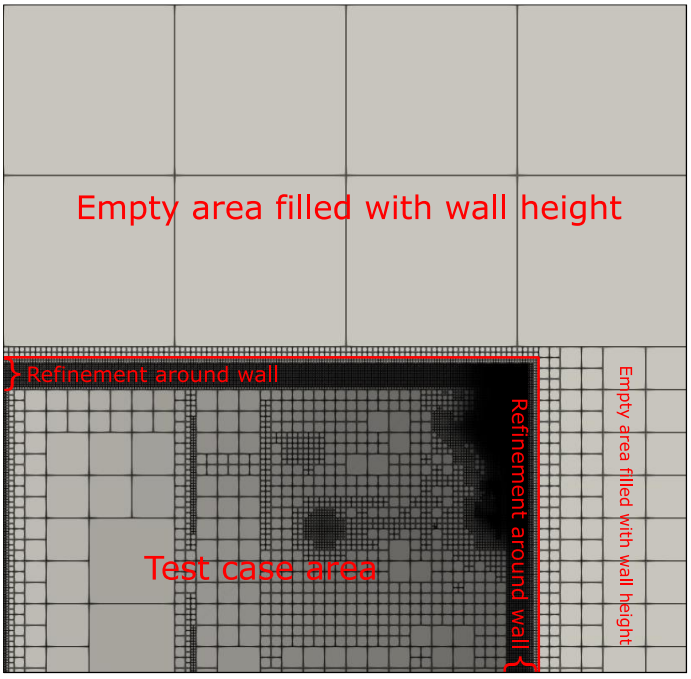

Figure 3: Initial non-uniform grid generated by the GPU-MWDG2 solver, via the MRA procedure, based on the time-invariant features the DEM for the "Monai Valley" test case (Sect. 3.1).

In the actual test case area, GPU-MWDG2 initialises the data in the cells by using the values specified in `.dem` and `.start` files in raster grid format (see Appendix B). Meanwhile, in the empty areas, it initialises the flow data to zero and assigns bathymetry data the numerical value that follows the `wall_height` keyword. This numerical value must be sufficiently high such that a high wall is generated between the test case area and the empty areas (see Figure 3) to prevents any water from leaving the test case area (e.g. by choosing a numerical value that is higher than the largest water surface elevation). For example, for the Monai valley test case, the `wall_height` keyword is specified to 0.5 m to generate a wall that is high enough to prevent any water from leaving the flow domain. The `refine_wall` and `ref_thickness` keywords, followed by an integer for the latter, typically between 16 and 64, should also be typed in the parameter file to prevent GPU-MWDG2 from excessively coarsening the non-uniform grid around the walls within the flow domain (labelled with the curly braces in Figure 3). This prevents very coarse cells in the empty areas, covered by user-added artificial topography, from being next to very fine cells in the actual areas, covered by actual DEM topography, that could otherwise cause unphysical predictions. For the Monai valley test case, the `refine_wall` keyword is specified to trigger refinement around the wall, and `ref_thickness` is specified as 16 to trigger 16 cells at the highest refinement level between the wall and the test case area.

The remaining keywords in Figure 2 are standard for running simulations using LISFLOOD-FP and were described previously[2]. Note that running GPU-MWDG2 on LISFLOOD-FP 8.2 only requires the user to provide the `.dem` file and

`.start` files—unlike the DG2 solvers in LISFLOOD-FP 8.0 (Shaw et al., 2021) and the static non-uniform grid generator in LISFLOOD-FP 8.1 (Sharifian et al., 2023), which require manual pre-processing of `.dem1x`, `.dem1y`, `.start1x` and `.start1y` raster files of initial DG2 slope coefficients. In the GPU-MWDG2 solver these coefficients are automatically pre-processed from the raw the `.dem` and `.start` files.

Compared to a GPU-DG2 simulation, a GPU-MWDG2 simulation consumes much more memory (Appendix A). As shown in Sect. 2.2, the large memory costs arise from the need to store the objects involved in the GPU-MWDG2 algorithm. Practically, the largest allowable choice of L or, in other words, the largest *square uniform grid* that can accommodate a DEM, is restricted by the memory capacity of the GPU card on which the GPU-MWDG2 simulation is performed.

2.2 GPU memory cost analysis, limits and scalability

The scope for running a GPU-MWDG2 simulation depends on the availability of a GPU card that can fit the memory costs for the specified choice of L. The left panel in Figure 4 shows the percentage breakdown of the memory consumed by the objects involved in GPU-MWDG2 simulations: the GPU-MWDG2 non-uniform grid, the explicitly deep-copied neighbours of each cell in the grid, and the hierarchy of grids involved in the dynamic GPU-MWDG2 adaptivity process (overviewed in Appendix A). It can be seen that 15% of the memory is allocated for arrays that store the shape coefficients of the cells in the hierarchy of uniform grids, while another 6% is allocated for other miscellaneous purposes. Remarkably however, nearly 80% of the overall GPU memory costs come from elsewhere: 22% from arrays storing the shape coefficients of the cells comprising the non-uniform grid, and 57% from arrays storing explicit deep-copies of the neighbouring shape coefficients of each cell (i.e. north, east, south, west). This high memory consumption using deep-copies ensures coalesced memory access when computing the DG2 solver updates (Appendix A). Furthermore, the GPU-MWDG2 solver is coded to allocate GPU memory for the worst-case scenario where there is no grid coarsening at all, thereby negating the need for memory reallocation after any coarsening to maximise the efficiency of dynamic GPU-MWDG2 adaptivity, since memory allocation is a relatively slow operation.

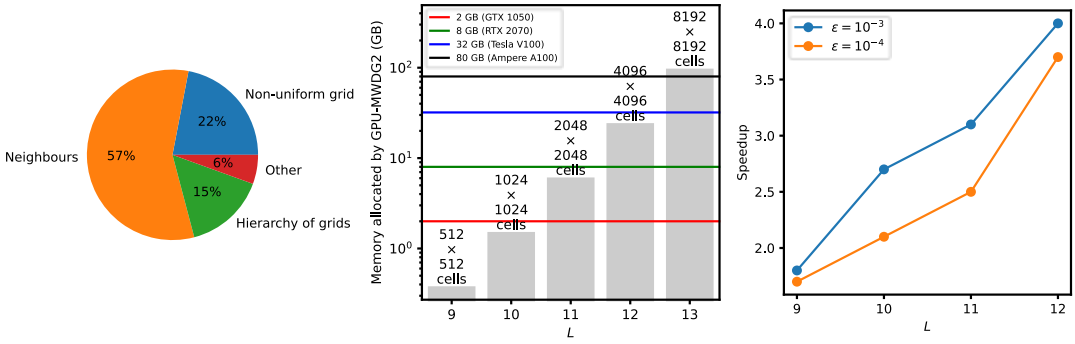

Figure 4: GPU memory consumed by dynamic GPU-MWDG2 adaptivity. Left panel shows the percentage breakdown of the GPU memory consumed by the different objects involved in the GPU-MWDG2 solver. Middle panel shows the amount of GPU memory

allocated against the maximum refinement level L; the numbers on top of the bars show the number of cells for a given value of L. The horizontal lines indicate the memory limits of four GPU cards. Right panel shows the scalability of the GPU-MWDG2 solver (i.e. an increasing speedup against an increasing workload) by running GPU-MWDG2 and GPU-DG2 simulations against increasing values of L for the "Monai valley" test case (Sect. 3.1) and plotting the speedups of the GPU-MWDG2 simulations compared to the GPU-DG2 simulations.

The middle panel in Figure 4 displays the GPU memory allocated by the GPU-MWDG2 simulations for different L leading to $2^L \times 2^L$ cells on the *square uniform grid*. The coloured lines represent the memory limits of four different GPU cards. In this figure, the memory limits are considered for $L \geq 9$, i.e. for the case where (multi)wavelet-based adaptive simulations were shown to start offering speedups over the uniform-grid simulations (A. A. Chowdhury et al., 2023; Kesserwani & Sharifian, 2023). As can be seen, dynamic GPU-MWDG2 adaptivity can only allocate GPU memory below the upper memory limit of the GPU card under consideration, leading to a restriction on the value of L that can be employed. For instance, a GTX 1050 card with a memory capacity of 2 GB can only accommodate GPU-MWDG2 simulations up to $L = 10$, i.e. starting from a *square uniform grid* made of 1024 × 1024 cells; this is because any value of $L > 10$ will lead to exceeding this GPU card's memory limit. Generally, the larger the value of L, the larger the $2^L \times 2^L$ cells on the *square uniform grid*, thus the larger the memory requirement for the GPU card. At the time this study was conducted, GPU-MWDG2 simulations involving $L \geq 13$, i.e. starting from a *square uniform grid* from 8192 × 8192 cells, were not feasible because accommodating such values of L needed >80 GB of GPU memory, which was higher than the memory limit of the latest commercially available GPU card (i.e. the A100 GPU card, with 80 GB of memory).

The right panel of Figure 4 shows speedups of GPU-MWDG2 simulations over GPU-DG2 simulations with respect to L for the "Monai valley" example (Sect. 3.1) to assess the scalability of the GPU-MWDG2 solver in relation to the simulated problem size (defined by L) for the selected A100 GPU card. It can be seen that the larger the simulated problem size, the higher and steeper the speedup. This indicates that the GPU-MWDG2 solver exhibits better scalability potential with increasing size of the simulated problem, such as coupled urban and river flooding over a very large area, requiring $L \geq 13$.

2.3 Proposed metrics for analysing GPU-MWDG's runtime efficiency

Assessing the speedup that could be afforded by GPU-MWDG2 adaptivity over a GPU-DG2 simulation is essential. As noted in other works that explored wavelet adaptivity, the computational effort and speedup of a GPU-MWDG2 simulation should ideally correlate with the number of cells in the GPU-MWDG2 non-uniform grid. This correlation is expected since the number of cells dictates the number of DG2 solver updates to be performed. However, this rarely occurs in practice as the ideal speedup is diminished by the additional computational effort spent by GPU-MWDG2 in generating the non-uniform grid every timestep via the MRA process (A. A. Chowdhury et al., 2023; Kesserwani et al., 2019; Kesserwani & Sharifian, 2020, 2023). To practically assess the potential speedup of a GPU-MWDG2 simulation, one must consider the

interdependent effects of the number of cells in the non-uniform grid, the computational effort of performing the DG2 solver updates, and the computational effort of performing the MRA process.

To do so, starting from LISFLOOD-FP 8.2, the user can include the "`cumulative`" keyword in the parameter file to produce a "`.cumu`" file that contains the time histories of several quantities for analysing the speedup achieved by GPU-MWDG2 adaptivity. This file contains time series of the number of cells in the non-uniform grid, the computational effort of performing the DG2 solver updates per timestep, the timestep size, the timestep count, amongst other items, with the full list of items detailed in the data and script files in A. A. Chowdhury & Kesserwani (2024). In this paper, the time histories of these quantities are postprocessed into several time-dependent metrics for analysing the speedups of GPU-MWDG2 simulations compared to GPU-DG2 simulations (Sect. 3). The metrics are described in Table 1, and their use for analysing the speedup of a GPU-MWDG2 simulation is explained next based on the Monai Valley example (Sect 3.1).

Table 1: Time-dependent metrics for evaluating the potential speedup of a GPU-MWDG2 simulation over a GPU-DG2 simulation.

Metric	Description
$N_{cells}(t)$	Number of cells in GPU-MWDG2's non-uniform grid compared to GPU-DG2's grid (as a percentage) against simulation time.
$R_{DG2}(t)$	Computational effort spent by GPU-MWDG2 to perform the DG2 solver updates at a given timestep (relative to GPU-DG2 as a percentage) against simulation time.
$R_{MRA}(t)$	Computational effort spent by GPU-MWDG2 to perform the MRA process and generate the non-uniform grid at a given timestep (relative to GPU-DG2 as a percentage) against simulation time.
$S_{inst}(t)$	Instantaneous speedup achieved by GPU-MWDG2 over GPU-DG2 at a given timestep against simulation time.
$N_{\Delta t}(t)$	Number of timesteps taken by GPU-MWDG2 to reach a given simulation time.
$C_{DG2}(t)$	Cumulative computational effort spent by GPU-MWDG2 to perform the DG2 solver updates (quantified in units of wall clock time) up to a given simulation time.
$C_{MRA}(t)$	Cumulative computational effort spent by GPU-MWDG2 to perform the MRA process (quantified in units of wall clock time) up to a given simulation time.
$C_{tot}(t)$	Total cumulative computational effort spent by GPU-MWDG2 to complete a simulation (quantified in units of wall clock time) up to a given simulation time.

$S_{acc}(t)$	Accumulated speedup of GPU-MWDG2 over GPU-DG2 up to a given simulation time.

In a GPU-MWDG2 simulation of an impact event, the computational effort per timestep changes depending on the change in the number of cells in the GPU-MWDG2 non-uniform grid. The number of cells changes over time because finer cells are generated by GPU-MWDG2 adaptivity to track the flow features produced by the impact event as it enters and travels through the bathymetric area. The left panel of Figure 5 shows the initial non-uniform grid generated by GPU-MWDG2 at the start of the simulation, while the right panel shows an intermediate non-uniform grid generated by GPU-MWDG2 after the simulation has progressed by 17 s, i.e. after an impact event, here a tsunami, has entered and propagated through the bathymetric area. At the start of the simulation, the initial non-uniform grid is coarsened as much as allowed, based only on the static features of the bathymetric area and initial flow conditions, leading to a minimal number of cells in the grid, which is quantified by N_{cell}. The number of cells determines the number of DG2 solver updates to be performed at a given timestep, leading to a corresponding computational effort per timestep, which is quantified by R_{DG2}. There is also the computational effort of performing the MRA process at a given timestep, which is quantified by R_{MRA}. Based on the combined computational effort of performing both the MRA process and the DG2 solver updates at a given timestep, the *instantaneous* speedup in completing one timestep of a GPU-MWDG2 simulation (relative to the GPU-DG2 simulation) can be computed, which is quantified by S_{inst}. In Figure 5, after the simulation has progressed by 17 s, the number of cells in the non-uniform grid has increased due to using finer cells to track the tsunami's wavefronts and wave diffractions, which leads to a higher value of N_{cell} and R_{DG2} (and possibly also to a higher value of R_{MRA}, as a higher number of cells in the non-uniform grid means more cells must be processed during the MRA process); thus, S_{inst} is expected to drop. Generally, the higher the complexity of the impact event, the higher the number of cells in the GPU-MWDG2 non-uniform grid, and the lower the potential speedup.

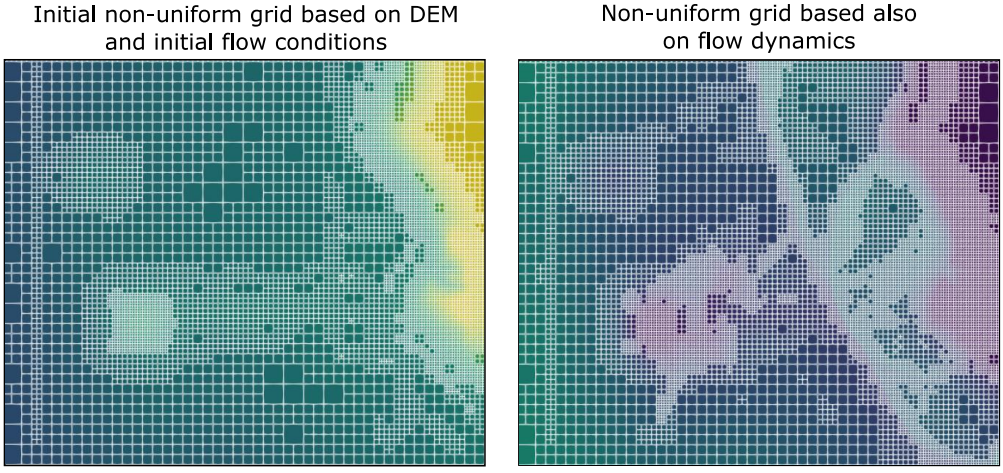

Figure 5: GPU-MWDG2 non-uniform grids generated for the "Monai Valley" test case (Sect. 3.1). Left panel shows the grid at the start of the simulation whereas the right panel shows the grid after the simulation has progressed by 17 s, which tracks flow dynamics.

The metrics N_{cell}, R_{DG2}, R_{MRA} and S_{inst} quantify the computational effort and speedup of a GPU-MWDG2 simulation *per timestep* compared to a GPU-DG2 simulation. However, the overall or *cumulative* computational effort and speedup of a GPU-MWDG2 simulation depends on having accumulated the computational effort and speedup per timestep from *all* the timesteps taken by GPU-MWDG2 to reach a given simulation time. The higher the number of timesteps taken by GPU-MWDG2 to reach a given simulation time (quantified by $N_{\Delta t}$), the higher the cumulative computational effort spent by GPU-MWDG2 to reach that simulation time (quantified by C_{tot}). The C_{tot} metric is computed by summing the cumulative computational effort spent by GPU-MWDG2 to perform the DG2 solver updates and the MRA process, which is quantified by C_{DG2} and C_{MRA}, respectively. Using the cumulative metrics, the overall speedup accumulated by a GPU-MWDG2 simulation can be computed, which is quantified by S_{acc}. The metrics in Table 1 are used in the next section (Sect. 3) to assess the speedup afforded by GPU-MWDG2 adaptivity over the GPU-DG2 simulation.

3 Evaluation of GPU-MWDG2 adaptivity

The efficiency of the GPU-MWDG2 solver is evaluated relative to the GPU-DG2 solver—which is always run on the DEM grid—using the metrics proposed in Table 1. The potential speedup afforded by GPU-MWDG2 adaptivity is hypothesised to depend on the duration and complexity of the impact event (Sect. 2.3) as well as the available DEM grid size, which dictates the choice for the L (Sect. 2.1). Therefore, the evaluation is performed by simulating four realistic test cases of tsunami-induced flooding with various impact event complexity, ranging from a single-wave to wave-train tsunamis, and DEM sizes, requiring L = 10 or 12 for the *square uniform grid*. The properties of the selected test cases are summarised in Table 2. The simulations were run on the Stanage high performance computing cluster of the University of Sheffield using an A100 GPU card with 80 GB of memory to accommodate the memory cost of using $L = 12$ (see Sect. 2.2).

The GPU-MWDG2 simulations are run with $\varepsilon = 10^{-4}$ and 10^{-3}, which is the range of ε where the predictive accuracy of the GPU-DG2 simulations can be preserved while achieving considerable speedups (Kesserwani & Sharifian, 2020, 2023). Here, the practical usability of these ε values is also validated based on the heuristic rule that ε should be proportional to Δx^L multiplied by a scaling factor C_ϵ (Caviedes-Voullième et al., 2020); where, Δx^L is the (finest) resolution of the DEM grid and C_ε represents the ratio between the smallest and largest length scale that can be captured during a simulation. For example, using $\varepsilon = 10^{-3}$ and 10^{-4} in the Monai Valley test case with the available grid resolution $\Delta x^L = 0.007$ yields $C_{10^{-3}} = 0.14$ and $C_{10^{-4}} = 0.014$, respectively, which are the highest length-scale ratios that can be captured in these simulations. Similarly, $C_{10^{-3}}$ and $C_{10^{-4}}$ were estimated for the other test cases based on the best available DEM resolutions, and they listed in Table 2.

Table 2: Information per test case. Size of $N \times M$ DEM grid and Manning coefficient n_M, size of *square uniform grid* to accommodate the DEM grid and the associated value of L, Δx^L the resolution of the DEM grid, C_ε the ratio between the smallest and largest length scale that can be captured in a simulation using a given value of ε and the Δx^L available, and the tsunami complexity.

Test case	DEM grid (n_M)	Square uniform grid (L)	$\Delta x^L[m]$	$C_{10^{-3}}$	$C_{10^{-4}}$	Tsunami complexity
"Monai Valley" (Sect. 3.1)	784 × 486 (0.01)	$2^{10} \times 2^{10}(10)$	0.007	0.14	0.014	Single wave
"Seaside Oregon" (Sect. 3.2)	2181 × 1091 (0.025)	$2^{12} \times 2^{12}(12)$	0.02	0.05	0.005	Single wave
"Tauranga Harbour" (Sect. 3.3)	4096 × 2196 (0.025)	$2^{12} \times 2^{12}(12)$	10	0.0001	0.00001	Low frequency wave train
"Hilo Harbour" (Sect. 3.4)	702 × 692 (0.025)	$2^{10} \times 2^{10}(10)$	10	0.0001	0.00001	High frequency wave train

According to Gerhard et al. (2015) and Gerhard & Müller (2016), a selected ε is valid if two conditions are met:

(i) the "full error" (FE) (the error between the GPU-MWDG2 predictions and the analytical solution) is in the same order of magnitude as the "discretisation error" (DE), i.e. the error between the GPU-DG2 predictions and the reference data; and

(ii) the "perturbation error" (PE) (the error between the GPU-MWDG2 and GPU-DG2 predictions) decreases by one order of magnitude when ε is also reduced by one order of magnitude for smooth numerical solutions.

In the present work, conditions (i) and (ii) can be approximately evaluated at observation points where measured data are available—to confirm the validity of $\varepsilon = 10^{-3}$ and 10^{-4}—by quantifying FE, DE and PE, which were calculated using the root mean squared error (RMSE):

$$FE = \frac{1}{N_{obs}} \sqrt{\Sigma_i^{N_{obs}} (P_i^{obs} - P_i^{MWDG2})^2} \quad (1)$$

$$DE = \frac{1}{N_{obs}} \sqrt{\Sigma_i^{N_{obs}} (P_i^{obs} - P_i^{DG2})^2} \quad (2)$$

$$PE = \frac{1}{N_{obs}} \sqrt{\Sigma_i^{N_{obs}} (P_i^{MWDG2} - P_i^{DG2})^2} \quad (3)$$

where N_{obs} is the number of observation points and P_i^{obs}, P_i^{DG2} and P_i^{MWDG2} are the i^{th} observation point, GPU-DG2 prediction and GPU-MWDG2 prediction respectively. The predictions may be any of the prognostic variables, to include water surface elevation $h + z$ or the u or v components of the velocity, or velocity-related variables.

3.1 Monai Valley

This test case has been used to validate many hydrodynamic solvers (Caviedes-Voullième et al., 2020; Kesserwani & Liang, 2012; Kesserwani & Sharifian, 2020; Matsuyama & Tanaka, 2001). It involves a 1:400 scaled replica of the 1993 tsunami that flooded Okushiri Island after a wave runup of 30 m at the tip of a very narrow gulley in a small cove at Monai Valley (Liu et al., 2008). The scaled DEM has 784 × 486 cells for which its associated initial *square uniform grid* is generated with $L = 10$.

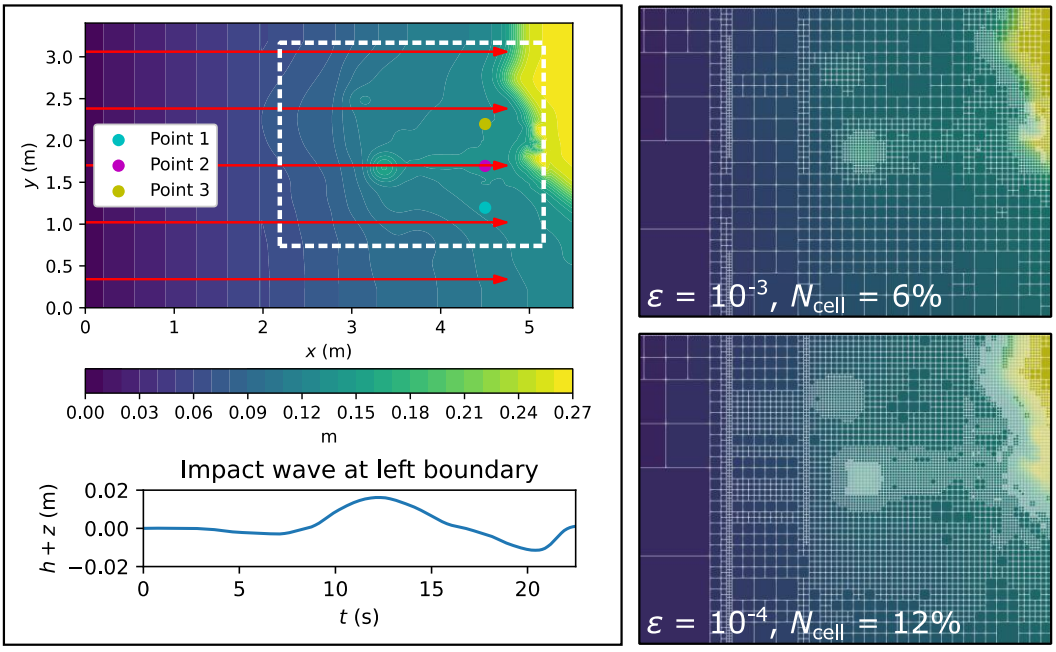

Figure 6: Monai Valley. Top-down view of bathymetry (top left panel), where the red arrows indicate the direction and distance travelled by the tsunami; time history of the tsunami entering from the left boundary (bottom left panel); initial GPU-MWDG2 non-uniform grids (right panels) covering the portion of the bathymetric area framed by the white box (top left panel).

In Figure 6, a top-down view of the bathymetric area is shown (top left panel), which has a small island in the middle and a coastal shoreline to the right, including Okushiri Island and Monai Valley. The coastal shoreline gets flooded by a tsunami that initially enters the bathymetric area from the left boundary and then travels to the right by 4.5 m (indicated by the red arrows), interacting with the small island as it travels through the bathymetric area. This tsunami is simulated for 22.5 s, during which it travels through the bathymetric area in three stages of flow over time: the entry stage (0 to 7 s), the travelling stage (7 to 17 s) and the flooding stage (17 to 22 s). During the entry stage, the tsunami does not enter the bathymetric area, as seen in the hydrograph of the tsunami's water surface elevation (bottom left panel of Figure 6). During the travelling stage, the tsunami enters from the left boundary and travels right towards the coastal shoreline. Lastly, during the flooding stage, the tsunami floods the coastal shoreline, due to which many flow dynamics such as wave reflections and diffractions are produced that must be tracked using finer cells, thus increasing the number of cells in the GPU-MWDG2

non-uniform grid. In the right panels of Figure 6, the initial GPU-MWDG2 grids at $\varepsilon = 10^{-3}$ and 10^{-4} are depicted for the portion of the bathymetric area framed by the white box (top left panel of Figure 6). With $\varepsilon = 10^{-3}$ and 10^{-4}, the initial GPU-MWDG2 grid has 6% and 12% of the number of cells as in the GPU-DG2 uniform grid, respectively.

In Figure 7, an analysis of the runtimes of the GPU-DG2 and GPU-MWDG2 simulations using the time-dependent metrics of Table 1 is shown. Up to 15 s, i.e. before the flooding stage of flow begins, the time history of N_{cell} remains flat, meaning that the number of cells in the GPU-MWDG2 non-uniform grid does not change over time. Once the flooding stage begins however, N_{cell} increases slightly, particularly at $\varepsilon = 10^{-4}$. With an increased number of cells in the non-uniform grid, the computational effort of performing the DG2 solver updates per timestep should increase, as is confirmed by the time history of R_{DG2}, which is flat before the flooding stage of flow, but thereafter increases, particularly at $\varepsilon = 10^{-4}$. However, unlike R_{DG2}, the time history of R_{MRA} is similar for both values of ε, and stays flat for most of the simulation except for an initial decrease at the start, meaning that the computational effort of performing the MRA process per timestep is similar for both values of ε and remains fixed throughout the simulation. Thus, the drop in the speedup of completing a single timestep of the GPU-MWDG2 simulation compared to the GPU-DG2 simulation is mostly due to the increase in R_{DG2} at $\varepsilon = 10^{-4}$, with S_{inst} dropping from 2.0 to 1.8 (which otherwise stays flat at 2.7 for $\varepsilon = 10^{-3}$).

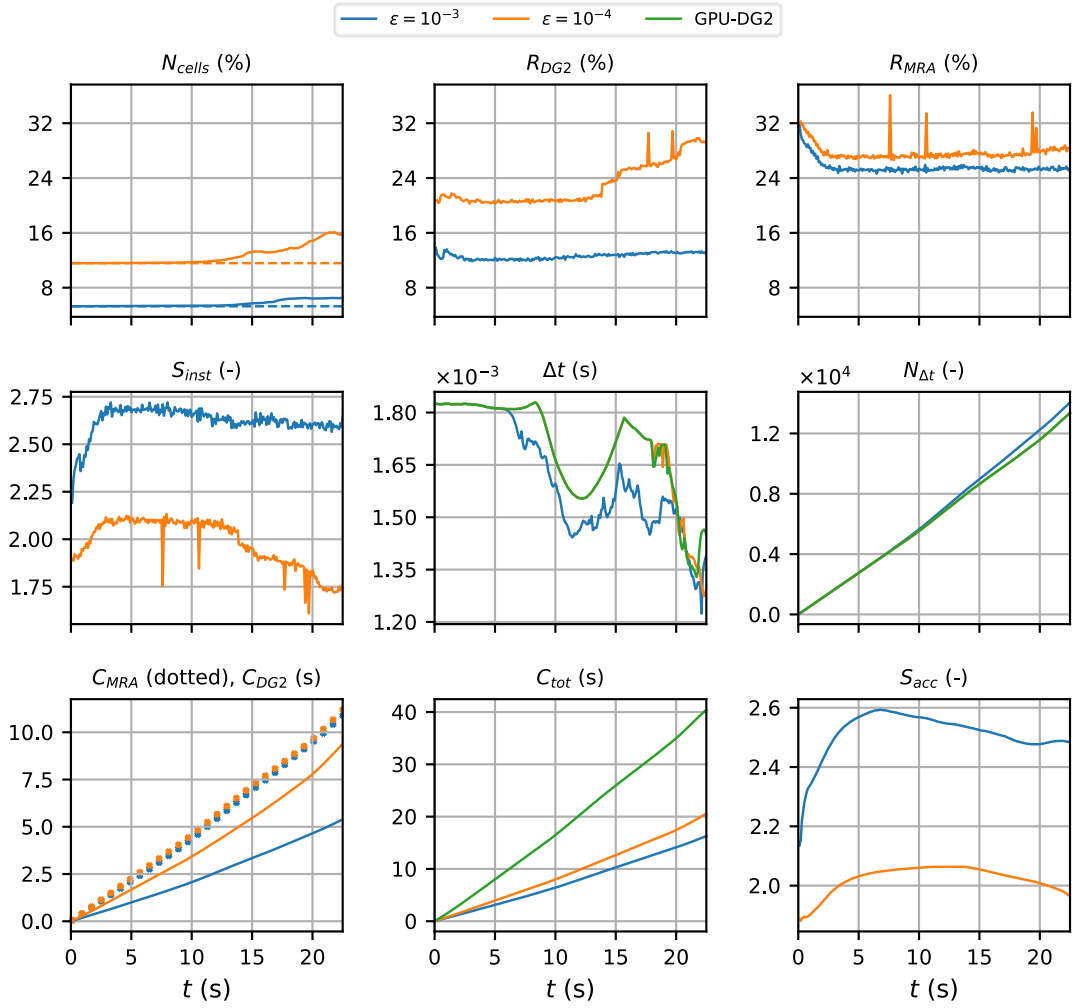

Figure 7: Monai Valley. Measuring the computational performance of the GPU-MWDG2 and GPU-DG2 simulations using the metrics of Table 1 to assess the speedup afforded by GPU-MWDG2 adaptivity: N_{cells},the number of cells in the GPU-MWDG2 non-uniform grid,where the dashed line represents the initial value at the beginning of the simulation; R_{DG2},the computational effort of performing the DG2 solver updates at given timestep relative to the GPU-DG2 simulation; R_{MRA}, the relative computational effort of performing the MRA process at a given timestep; S_{inst}, the instantaneous speedup achieved by the GPU-MWDG2 simulation over the GPU-DG2 simulation at a given timestep; Δt, the simulation timestep; $N_{\Delta t}$, the number of timesteps taken to reach a given simulation time; C_{DG2}, the cumulative computational effort of performing the DG2 solver updates up to a given simulation time; C_{MRA}, the cumulative computational effort of performing the MRA process; C_{tot}, the total cumulative computational effort; S_{acc}, the accumulated speedup of GPU-MWDG2 over GPU-DG2 up to a given simulation time.

The cumulative computational effort is affected by the timestep size (Δt) and the number of timesteps taken to reach a given simulation time ($N_{\Delta t}$). In this test case, the time histories of Δt of the GPU-DG2 simulation and the GPU-MWDG2 simulation using $\varepsilon = 10^{-4}$ are very similar, but with $\varepsilon = 10^{-3}$, it drops at 7 s, i.e. as soon as the travelling stage of flow begins. This slight drop in Δt is likely caused by the existence of planar DG2 solutions per cell with either partially wet portions or thin water layers under friction effects that locally and temporarily triggers smaller-scale timesteps. Either of

these are more likely to be present in coarser cells, thereby reducing Δt with $\varepsilon = 10^{-3}$. Due to the smaller Δt at $\varepsilon = 10^{-3}$, the trend in $N_{\Delta t}$ is steeper at $\varepsilon = 10^{-3}$ than 10^{-4}, i.e. more timesteps are taken and thus more computational effort is spent by GPU-MWDG2 to reach a given simulation time at $\varepsilon = 10^{-3}$ than 10^{-4}. Still, despite the steeper trend in $N_{\Delta t}$, the cumulative computational effort of performing the MRA process is similar using both $\varepsilon = 10^{-3}$ and 10^{-4}, which is expected given that R_{MRA} is also very similar for values of ε. In contrast, the cumulative computational effort of performing the DG2 solver updates is considerably higher at $\varepsilon = 10^{-4}$ than 10^{-3}, likely due to the higher R_{DG2} at $\varepsilon = 10^{-4}$, which seems correct since a higher computational effort to perform the DG2 solver updates per timestep should lead to a higher cumulative computational effort (assuming that the time histories of $N_{\Delta t}$ are similar for the different values of ε, which is the case here). Notably though, C_{DG2} is actually smaller than C_{MRA}, meaning that in this test case, the computational effort is dominated by the MRA process rather than the DG2 solver updates. Overall, for both values of ε, the total computational effort of running the GPU-MWDG2 simulations is always lower than that of the GPU-DG2 simulation, with C_{tot} always being lower than that of GPU-DG2 at both $\varepsilon = 10^{-3}$ and 10^{-4}. The accumulated speedups of the GPU-MWDG2 simulations, S_{acc}, finish at around 2.5 and 2.0 using $\varepsilon = 10^{-3}$ and 10^{-4}, respectively.

In Figure 8, the GPU-MWDG2 predictions at $\varepsilon = 10^{-3}$ and 10^{-4} and the GPU-DG2 prediction of the water surface elevation $h + z$ at points 1, 2 and 3 (i.e. the coloured points in the top left panel of Figure 7) are shown. All three predictions are very similar to each other visually, although at point 2, the wave arrival time is seen to be slightly underpredicted using $\varepsilon = 10^{-3}$. In Table 3, the FE, DE and PE at the same points are shown: the FE and DE are of the same order of magnitude, while the PE decreases by around an order of magnitude since the simulation does not involve strong discontinuities. This shows fulfilment of conditions (i) and (ii), in turn confirming the validity of the selected ε values in this test case.

Overall, the GPU-MWDG2 solver closely reproduces the GPU-DG2 water surface elevation predictions while being two times faster than the GPU-DG2 solver due to using $L = 10$. This is in line with the speedups shown in Figure 4 in Sect 2.2, which suggests that $L = 10$ is the "borderline" value of L where GPU-MWDG2 adaptivity reliably yields a speedup. In the next test case, the impact of a larger DEM size—requiring a larger L value—on the speedup of the GPU-MWDG2 solver is evaluated. The prediction of more complex velocity-related quantities is also considered.

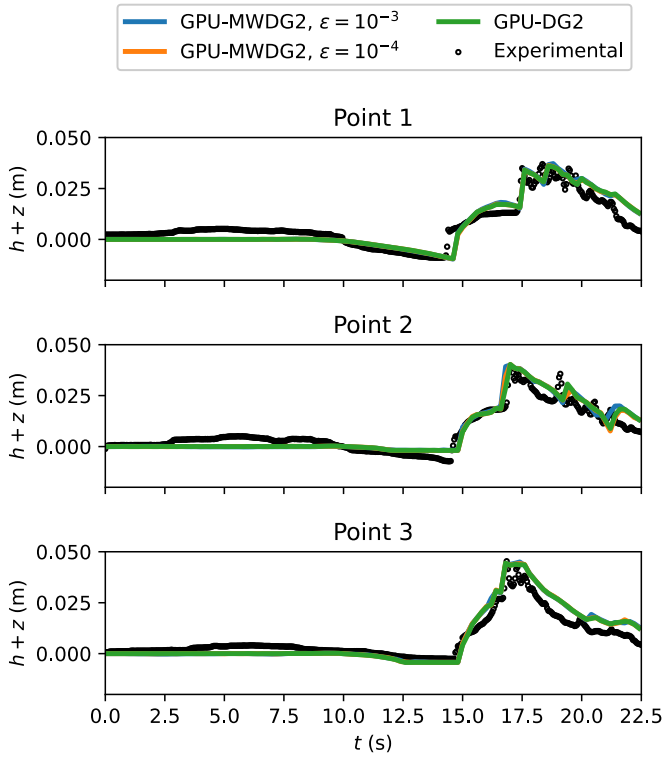

Figure 8: Monai Valley. Time series of the water surface elevation ($h + z$) predicted by GPU-DG2 and GPU-MWDG2 at the three sampling points (shown in Figure 6, top left panel) compared to the experimental results.

Table 3: Monai valley. DE, FE and PE quantified using Eqs. (1) to (3) considering measured and predicted time series of the water surface elevation at points 1, 2 and 3 in Figure 7. The errors are computed in order to check for fulfilment of conditions (i) and (ii) and to confirm the validity of selecting $\varepsilon = 10^{-3}$ and 10^{-4} in this test case.

	DE	**FE**		**PE**	
Observation point	-	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$
Point 1	0.012429	0.012544	0.012414	0.000814	0.000218
Point 2	0.004829	0.005277	0.004818	0.001606	0.000767
Point 3	0.007427	0.007690	0.007323	0.001209	0.000474

3.2 Seaside Oregon

This is another popular benchmark test case that has been used to validate hydrodynamic solvers for nearshore tsunami inundation simulations (Gao et al., 2020; Macías, Castro, & Escalante, 2020; H. Park et al., 2013; Qin et al., 2018; Violeau et

al., 2016). It involves a 1:50 scaled replica of an urban town in Seaside, Oregon that is flooded by a tsunami travelling along a scaled DEM made up of 2181 × 1091 cells, here requiring a larger $L = 12$ to generate the initial *square uniform grid*.

In Figure 9, the bathymetric area is shown (top left panel), which is very flat everywhere except to the right where very complex terrain features of the urban town, such as buildings and streets, are located. The urban town is flooded by a tsunami that enters from the left boundary and travels a distance of 33 m to the right before flooding the town (as shown by the red arrows). This tsunami is simulated for 40 s, during which it travels through the bathymetric area in four stages of flow over time, much like in the last test case (Sect. 3.1): the entry stage (0 to 10 s), the travelling stage (10 to 25 s), the flooding stage (25 to 35 s) and the inundation stage (35 to 40 s). During the entry stage, the tsunami is not yet in the bathymetric area; during the travelling stage, the tsunami starts to enter the bathymetric area from the left boundary and travels right towards the town; during the flooding stage, the tsunami hits the town, flooding the streets and overtopping some of the buildings, causing vigorous flow dynamics; finally, during the inundation stage, the tsunami inundates the town and goes on to interact with the right boundary, causing wave reflections. The water surface elevation hydrograph of the tsunami is plotted in the bottom left panel of Figure 9: it only has a single peak, therefore leading to low tsunami complexity. The right panels show the initial GPU-MWDG2 non-uniform grids at $\varepsilon = 10^{-3}$ and 10^{-4}, respectively (again for the portion of the bathymetric area framed by the white box in top left panel): at $\varepsilon = 10^{-3}$, greater grid coarsening is achieved, with the grid including only 2% of the number of cells as in the GPU-DG2 uniform grid, whereas at $\varepsilon = 10^{-4}$ it is 5% since more cells are used due to retention of finer resolution around and within complex terrain features of the urban town.

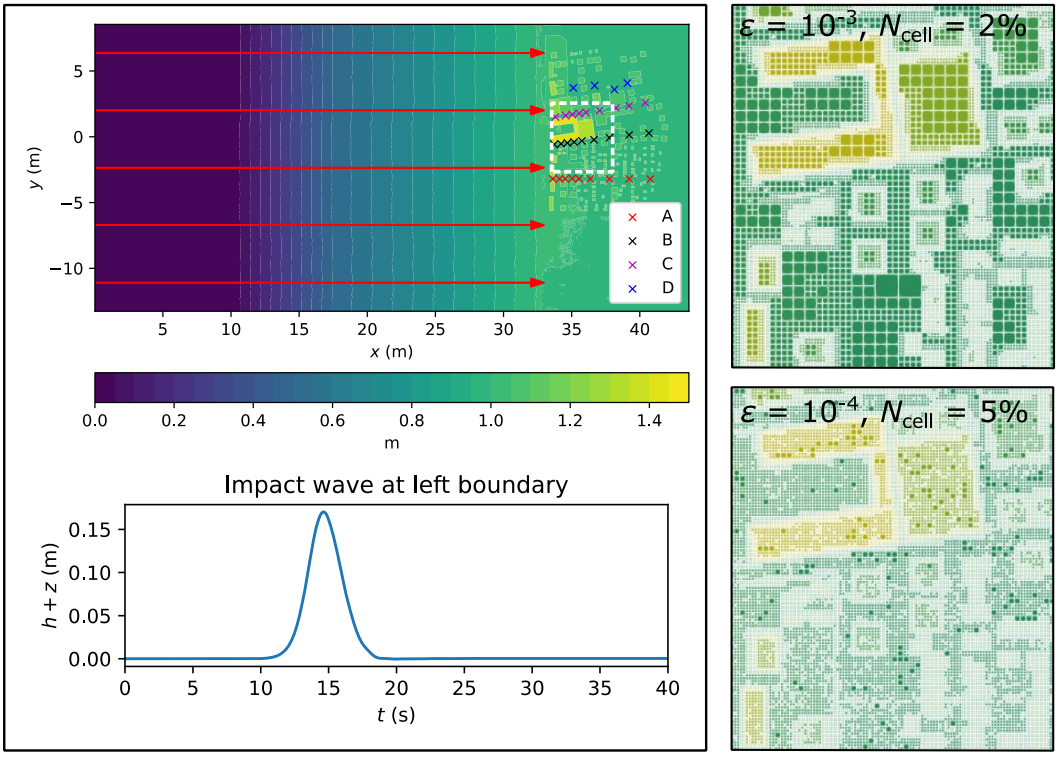

Figure 9: Seaside Oregon. Top-down view of bathymetry (top left panel), where the red arrows indicate the direction and distance travelled; tsunami time history entering the left boundary (bottom left panel); initial GPU-MWDG2 grids (right panels) for the potion in white box (top left panel).

In Figure 10, an analysis of runtimes of the GPU-MWDG2 and GPU-DG2 simulations are shown. As indicated by the time history of N_{cell}, the number of cells in the GPU-MWDG2 non-uniform grid does not increase significantly for either value of ε until the flooding stage of flow at 25 s, where N_{cell} starts increasing more noticeably. Once the number of cells starts increasing, there is a corresponding increase in R_{DG2}. On the other hand, the time history of R_{MRA} is quite flat during the entire simulation, except for a small decrease during the first 10 s of the simulation, and a small increasing trend in the final 5 s of the simulation, i.e. during the inundation stage of flow when the number of cells increases relatively sharply compared to the rest of the simulation. Driven primarily by the increase in R_{DG2} at the flooding stage at 25 s, the time history of S_{inst} is quite stable until 25 s and thereafter shows a decreasing trend that is particularly steep at $\varepsilon = 10^{-3}$.

Like the previous test case (Sect. 3.1), the time histories of Δt in the GPU-DG2 simulation and GPU-MWDG2 simulation at $\varepsilon = 10^{-4}$ are very similar, but at $\varepsilon = 10^{-3}$, there is a sharp drop in Δt after 32 s, i.e. when the flooding stage of flow starts transitioning to the inundation stage. Again, this sharp drop in Δt is triggered by coarse cells that are either partially wet or have thin water layers, which are more likely in the non-uniform grid with $\varepsilon = 10^{-3}$, but not with 10^{-4}. The first drop in Δt, which occurs at 25 s when the flooding stage starts, leads to a locally steeper trend in the time history of $N_{\Delta t}$, as indicated by the kink in the trend at 25 s. The second drop in Δt, which is seen only for $\varepsilon = 10^{-3}$ after 32 s, leads to a sustained steepness in the time history of $N_{\Delta t}$. This steepness means that GPU-MWDG2 takes more timesteps and thus accumulates more computational effort to reach a given simulation time at $\varepsilon = 10^{-3}$ than 10^{-4}, which is confirmed by the final value of C_{MRA}, which is higher at the end of the simulation at $\varepsilon = 10^{-3}$ compared to 10^{-4}, even though it was lower at $\varepsilon = 10^{-3}$ than at $\varepsilon = 10^{-4}$ for the rest of the simulation. Since R_{MRA} is always lower at $\varepsilon = 10^{-3}$ than 10^{-4}, this observation about C_{MRA} suggests that even if the computational effort per timestep is lower throughout the simulation, a high timestep count can sufficiently increase the cumulative computational effort such that it becomes higher at $\varepsilon = 10^{-3}$ than 10^{-4}. Nonetheless, the time history of C_{tot} in the GPU-MWDG2 simulations always remains well below that of the GPU-DG2 simulation, with S_{inst} finishing at 3.5 and 3.0 with $\varepsilon = 10^{-3}$ and 10^{-4}, respectively.

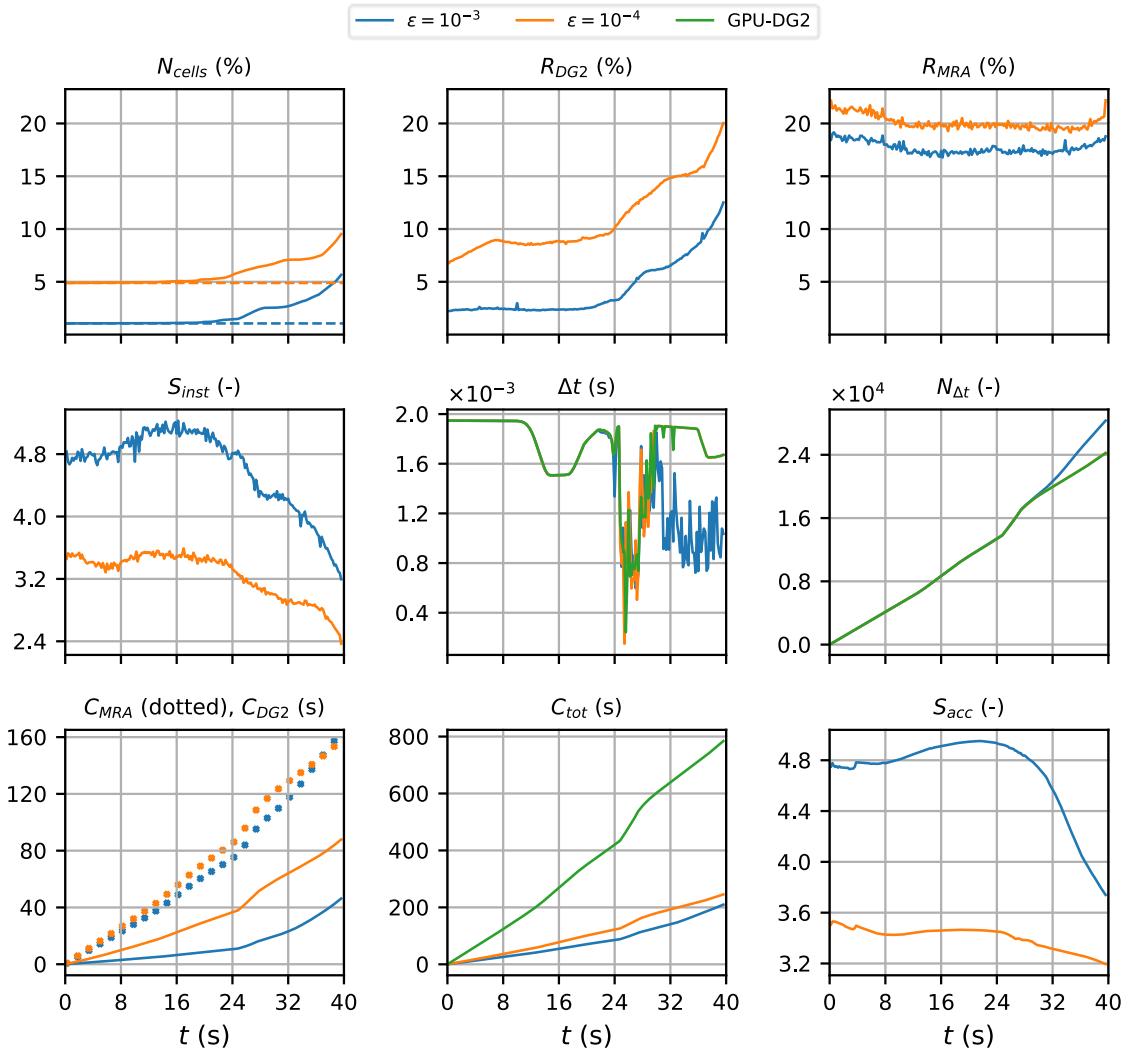

Figure 10: Seaside Oregon. Measuring the computational performance of the GPU-MWDG2 and GPU-DG2 simulations using the metrics of Table 1 to assess the speedup afforded by GPU-MWDG2 adaptivity: N_{cells},the number of cells in the GPU-MWDG2 non-uniform grid,where the dashed line represents the initial value at the beginning of the simulation; R_{DG2},the computational effort of performing the DG2 solver updates at given timestep relative to the GPU-DG2 simulation; R_{MRA}, the relative computational effort of performing the MRA process at a given timestep; S_{inst}, the instantaneous speedup achieved by the GPU-MWDG2 simulation over the GPU-DG2 simulation at a given timestep; Δt, the simulation timestep; $N_{\Delta t}$, the number of timesteps taken to reach a given simulation time; C_{DG2}, the cumulative computational effort of performing the DG2 solver updates up to a given simulation time; C_{MRA}, the cumulative computational effort of performing the MRA process; C_{tot}, the total cumulative computational effort; S_{acc}, the accumulated speedup of GPU-MWDG2 over GPU-DG2 up to a given simulation time.

In Figure 11, the time series predicted by GPU-MWDG2 using ε = 10^{-3} and 10^{-4} and GPU-DG2 of the prognostic variables $h + z$, the u component of the velocity, and the derived momentum variable $M_x = 0.5hu^2$ at points A1, B6 and D4 are shown. Point A1 is one of the left-most crosses in Figure 9, point B6 is one of the central crosses, and point D4 is one of the right-most crosses. At all three points, the predictions are visually similar to each other except at point D4, where, for ε =

10^{-3}, the impact wave height is slightly overpredicted, while the velocity and momentum is overpredicted between 33 s and 43 s. Compared to the measured data, the predictions all follow the trailing part of the measurement time series quite well, but at the peaks, they are underpredicted at point A1, overpredicted at point B6, and out of phase at point D4. These differences, which manifest most noticeably at the peaks, may be due to the incoming bore of the impact wave, which had invalidated measurements and required repeating experiments, possibly introducing uncertainty into the measured data (H. Park et al., 2013).

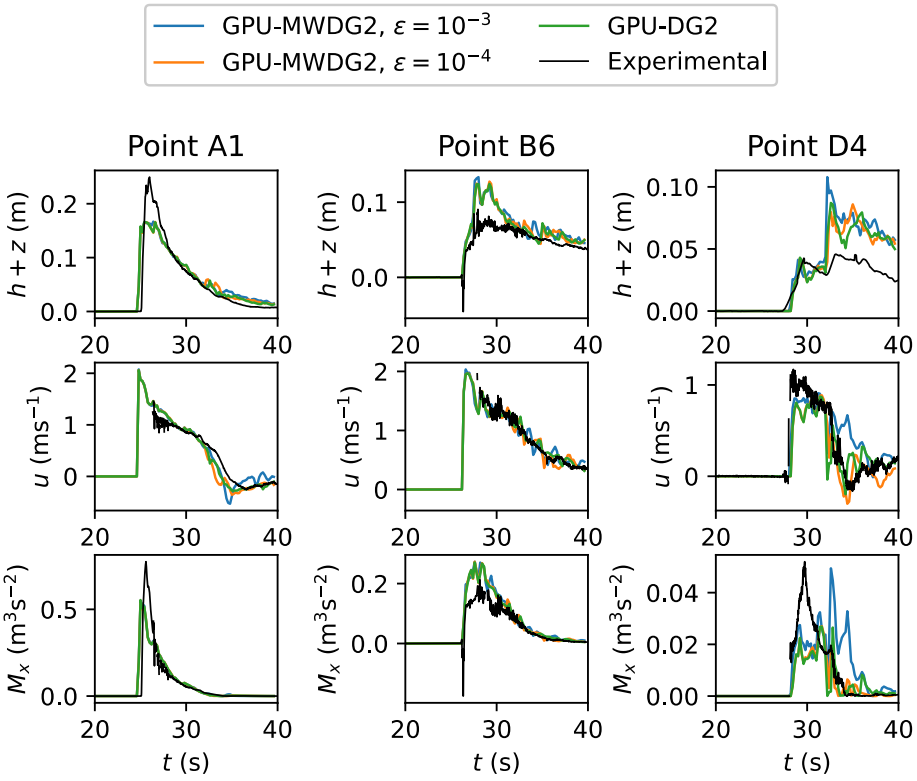

Figure 11: Seaside Oregon. Time series of the water surface elevation ($h + z$), u velocity component and momentum M_x for the GPU-DG2 and GPU-MWDG2 predictions at points A1, B6 and D4 (Figure 9), compared to the experimental data.

Table 4 shows the associated DE, FE and PE at points A1, B6 and D4. The errors confirm fulfilment of conditions (i) and (ii) and thus show that selected ε values are valid: the DE and FE are in the same order of magnitude, while decreasing ε from 10^{-3} to 10^{-4} leads to a reduction in the PE—although the reduction is less than an order of magnitude compared to the last test case (Sect. 3.1) due to the strong discontinuities introduced into the numerical solution by the buildings in the urban area.

Table 4: Seaside Oregon. DE, FE and PE quantified using Eqs. (1) to (3) considering measured and predicted time series of the prognostic variables $h + z$, u and M_x at points A1, B6 and D4 (shown in Figure 9). The errors are considered in order to check for fulfilment of conditions (i) and (ii) and confirm the validity of selecting $\varepsilon = 10^{-3}$ and 10^{-4} in this test case.

		DE	FE		PE	
Observation point	**Quantity**	-	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$
A1	$h + z$	0.168783	0.168835	0.168764	0.003100	0.001857
	u	0.926009	0.928524	0.925210	0.057247	0.026415
	M_x	0.316526	0.316487	0.316498	0.002713	0.000809
B6	$h + z$	0.065732	0.065925	0.065724	0.004869	0.003087
	u	1.597801	1.598185	1.597701	0.060968	0.042121
	M_x	0.148552	0.148858	0.148491	0.007611	0.003425
D4	$h + z$	0.013469	0.014846	0.012927	0.007300	0.004517
	u	0.145877	0.169860	0.160767	0.194791	0.121198
	M_x	0.005891	0.007861	0.006407	0.007534	0.003531

This test case and the previous show that the GPU-MWDG2 solver can achieve at least a 2-fold speedup over the GPU-DG2 solver for tsunami simulations involving a single-wave impact event. Notably though, if the impact event had been present for a larger fraction of the simulation time, it would have introduced a correspondingly higher amount of complexity over time, and a higher cumulative computational effort would have been expected. This type of event will be explored next in Sects. 3.3 and 3.4.

3.3 Tauranga Harbour

This test case reproduces the 2011 Japan tsunami event in Tauranga Harbour, New Zealand (Borrero et al., 2015; Macías, Castro, Ortega, et al., 2020; Macías et al., 2015). The bathymetric area has a DEM made of 4096 × 2196 cells, requiring L = 12 to generate the initial *square uniform grid*. As shown by the red arrows in Figure 12, the tsunami enters from the top boundary and travels a short distance downwards before quickly hitting the coast at y = 16 km. As shown by the time history of the water surface elevation (bottom left panel of Figure 12), the tsunami is a wave train made up of three wave peaks and troughs that enter that bathymetric area one after the other at 0, 12 and 24 hr during the 40-hr tsunami event, with the latter two waves also exhibiting noise. Vigorous flow dynamics occur within the bathymetric area from the very beginning of the simulation, due to complex impact event and the irregular bathymetric zones that trigger wave reflections and diffractions. The right panels of Figure 6 include the GPU-MWDG2 grids generated at $\varepsilon = 10^{-3}$ and 10^{-4} for the region bounded by the white box (top left panel). With $\varepsilon = 10^{-3}$, the number of cells in the grid is 5% of the GPU-DG2 uniform grid, whereas with $\varepsilon = 10^{-4}$, it is 15%, due to less coarsening in and around the irregular bathymetric zones.

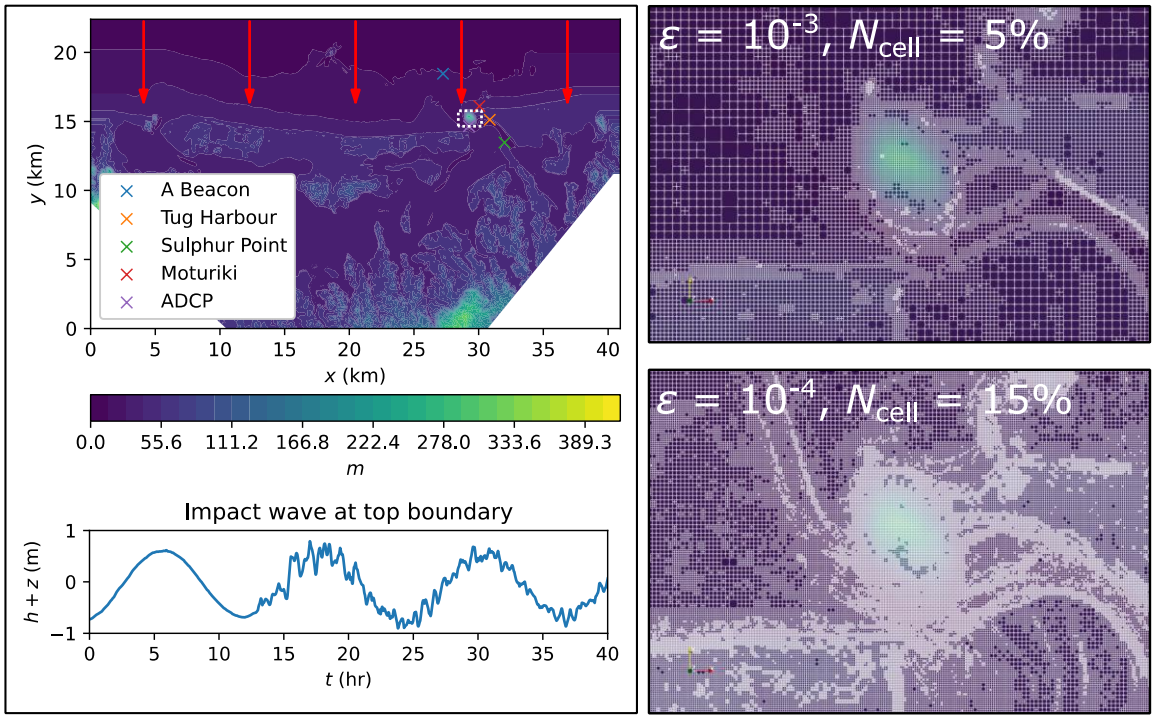

Figure 12: Tauranga Harbour. Top-down view of bathymetry (top left panel), where the red arrows indicate the direction and distance travelled; tsunami time history entering the top boundary (bottom left panel); initial GPU-MWDG2 grids (right panels) for the potion in white box (top left panel).

In Figure 13, an analysis of the runtimes of the GPU-MWDG2 and GPU-DG2 simulations is shown. Unlike the previous test cases (Sects. 3.1 and 3.2), the first wave of the tsunami wave train enters the bathymetric immediately, causing N_{cell} to increase very sharply and immediately from its initial value for both values of ε, which thereafter fluctuates due to the periodic tsunami signal. Following the sharp increase and fluctuations in N_{cell}, R_{DG2} also sharply increases and fluctuates. However, R_{MRA} does not and stays stable and flat throughout the simulation. Thus, driven primarily by the sharp decrease in R_{DG2}, S_{inst} decreases sharply from 4.0 to 1.6. The time histories of Δt in the GPU-DG2 simulation and the GPU-MWDG2 simulation using $\varepsilon = 10^{-4}$ follow each other quite closely up to 16 h, but at $\varepsilon = 10^{-3}$, the time history of Δt shows two periodic drops after 24 h. These drops are likely due to periodic wetting and drying processes that trigger coarse cells with partially wet portions or thin water layers, which are more common in the non-uniform grid at $\varepsilon = 10^{-3}$ but not at 10^{-4}. Due to the smaller Δt at $\varepsilon = 10^{-3}$, the time history of $N_{\Delta t}$ is locally steeper (see the kinks at 25 and 35 h), but this does not lead to significant differences between the cumulative computational effort at $\varepsilon = 10^{-3}$ versus 10^{-4}. The time history of C_{tot} in the GPU-MWDG2 simulations remains consistently below that of the GPU-DG2 simulation for both values of ε, but they are relatively close to each other compared to the previous test cases (Sects. 3.1 and 3.2). Thus, even though S_{acc} starts at around 4, like in the previous test case with $L = 12$ (Sect. 3.2), it drops sharply to 1.6 and 1.4 at $\varepsilon = 10^{-3}$ and 10^{-4}, respectively.

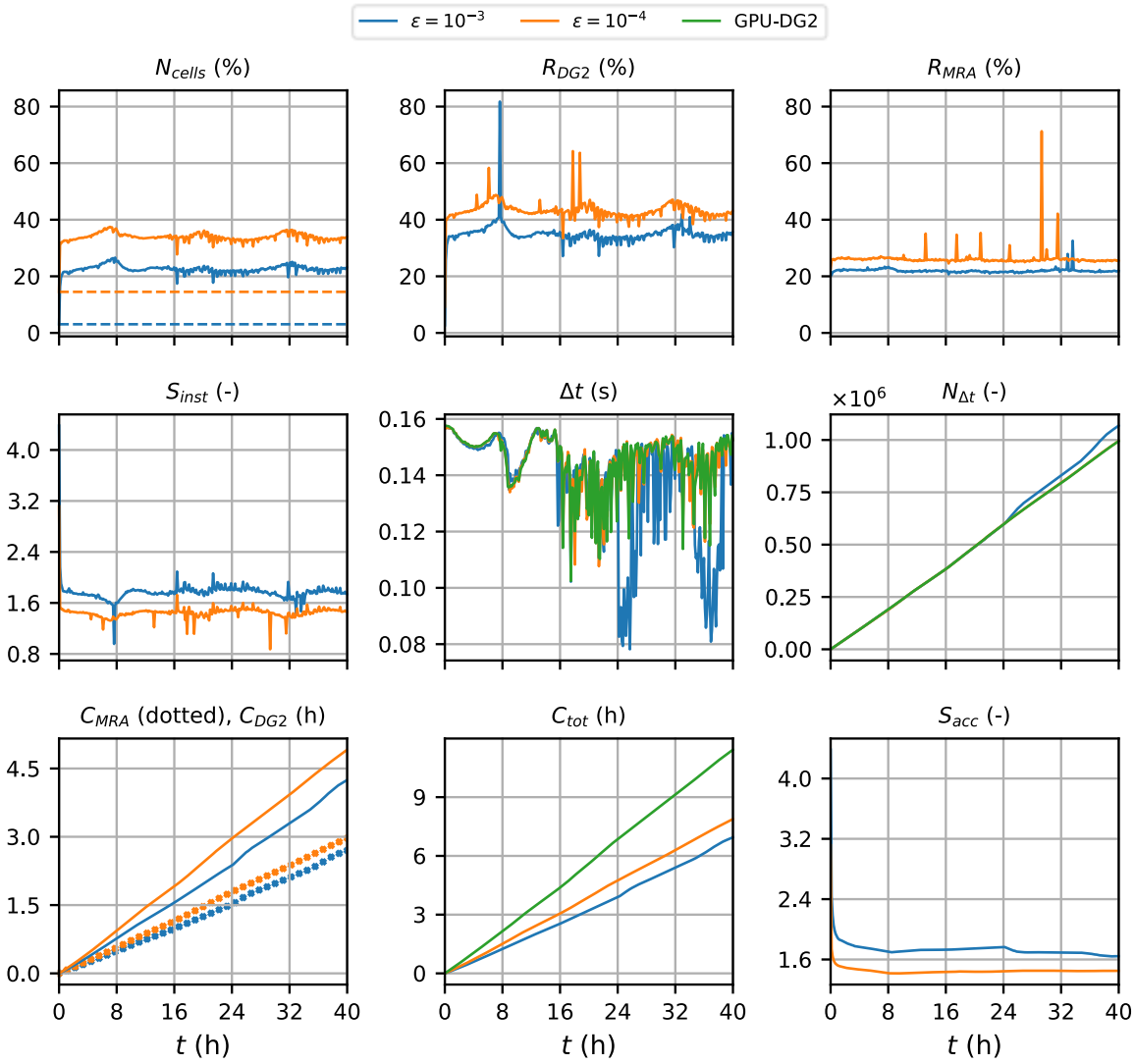

Figure 13: Tauranga Harbour. Measuring the computational performance of the GPU-MWDG2 and GPU-DG2 simulations using the metrics of Table 1 to assess the speedup afforded by GPU-MWDG2 adaptivity: N_{cells},the number of cells in the GPU-MWDG2 non-uniform grid,where the dashed line represents the initial value at the beginning of the simulation; R_{DG2},the computational effort of performing the DG2 solver updates at given timestep relative to the GPU-DG2 simulation; R_{MRA}, the relative computational effort of performing the MRA process at a given timestep; S_{inst}, the instantaneous speedup achieved by the GPU-MWDG2 simulation over the GPU-DG2 simulation at a given timestep; Δt, the simulation timestep; $N_{\Delta t}$, the number of timesteps taken to reach a given simulation time; C_{DG2}, the cumulative computational effort of performing the DG2 solver updates up to a given simulation time; C_{MRA}, the cumulative computational effort of performing the MRA process; C_{tot}, the total cumulative computational effort; S_{acc}, the accumulated speedup of GPU-MWDG2 over GPU-DG2 up to a given simulation time.

In Figure 14, the top four panels show the time series of the water surface elevation predicted by GPU-MWDG2 using $\varepsilon = 10^{-3}$ and 10^{-4} and by GPU-DG2 at observation points A Beacon, Tug Harbour, Sulphur Point and Moturiki (top left panel of Figure 12). All of the predictions are visually in agreement with each other as well as the measured data.

Meanwhile, the bottom panel of Figure 14 shows the speed ($\sqrt{u^2 + v^2}$) predicted at observation point ADCP, where the predictions are still visually in agreement with each other, but less so with the measured speed, as was also observed for the velocity-related predictions in the last test case (Sect. 3.2). Table 5 quantifies the extent of agreement by showing the associated DE, FE and PE at the observation points. The DE and FE are in the same order of magnitude, while the PE is reduced by close to an order of magnitude when ε is decreased from 10^{-3} to 10^{-4}, fulfilling conditions (i) and (ii) and thereby validating the used ε choices.

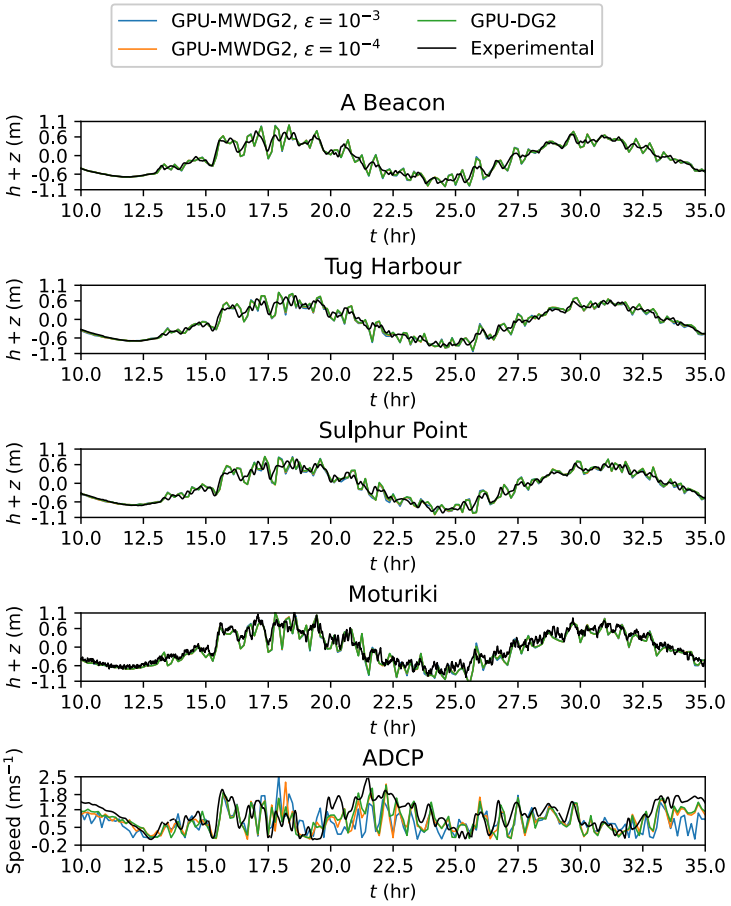

Figure 14: Tauranga Harbour. Time series of the water surface elevation ($h + z$) produced by GPU-DG2 and GPU-MWDG2 at the points labelled A Beacon, Tug Harbour, Sulphur Point and Moturiki (labelled in Figure 12) and for the speed ($\sqrt{u^2 + v^2}$) at the point ADCP (also labelled in Figure 12), compared to the experimental results.

Overall, this test case features a more complex tsunami compared to the previous test cases (Sects. 3.1 and 3.2), which sharply increases the number of cells in the GPU-MWDG2 non-uniform grid and thus also increases the computational effort of performing the DG2 solver updates. Hence, despite requiring the same $L = 12$ as the previous test case (Sect. 3.2), the final speedups are lower in this test case due to the more complex tsunami, with S_{acc} finishing at 1.6-

and 1.4-fold with $\varepsilon = 10^{-3}$ and 10^{-4}, respectively. Both choices of $\varepsilon = 10^{-4}$ and 10^{-3} are valid for this test case: using $\varepsilon = 10^{-4}$ would improve the closeness to the GPU-DG2 predicted velocities, while using $\varepsilon = 10^{-3}$ leads to very close water surface elevation predictions and fairly accurate velocity predictions, although without a major improvement in the speedup. In the next test case, another complex tsunami with higher frequency impact event peaks is considered, but now with a smaller DEM size requiring $L = 10$.

Table 5: Tauranga Harbour. DE, FE and PE quantified using Eqs. (1) to (3) considering measured and predicted time series of the water surface elevation $h + z$ at observation points A Beacon, Tug Harbour, Sulphur Point and Moturiki (top left panel of Figure 12), and the speed ($\sqrt{u^2 + v^2}$) at observation point ADCP. The errors are considered in order to check for fulfilment of conditions (i) and (ii) and confirm the validity of selecting $\varepsilon = 10^{-3}$ and 10^{-4} in this test case.

		DE	**FE**		**PE**	
Observation point	**Quantity**		$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$
A Beacon	$h + z$	0.150932	0.153826	0.151580	0.013583	0.002991
Tug Harbour	$h + z$	0.155685	0.157995	0.157293	0.032124	0.008822
Sulphur Point	$h + z$	0.154380	0.162152	0.154730	0.040685	0.009663
Moturiki	$h + z$	0.225445	0.225461	0.225444	0.028375	0.008321
ACDP	$Speed$	0.150932	0.153826	0.151580	0.013583	0.002991

3.4 Hilo Harbour

This test case reproduces the 2011 Japan tsunami event at Hilo Harbour in Hawaii, USA (Arcos & LeVeque, 2014; Lynett et al., 2017; Macías, Castro, Ortega, et al., 2020; Velioglu Sogut & Yalciner, 2019). It involves a complex tsunami made up of a high-frequency wave train that propagates for 6 hr into a bathymetric area that is smaller than the previous test case (Sect. 3.3): the latter bathymetric area has a DEM size made of 702 × 692 cells, requiring a smaller $L = 10$ to generate the initial *square uniform grid*. As shown in Figure 15, the tsunami enters from the top boundary and travels south to flood the coast at $y = 4$ km. The wave train occurs over the entire 6 hr simulation time, from a reference timestamp of 7 hr to 13 hr post-earthquake. In Figure 15, the time history of the wave train is shown during the 8.5 to 11 hr time period (bottom left panel): from the very beginning and during the entire simulation, violent flow dynamics occur in the bathymetric area. The right panels show the initial GPU-MWDG2 non-uniform grids generated at $\varepsilon = 10^{-3}$ and 10^{-4} for the region bounded by the white box (top left panel): the number of cells in the initial non-uniform grids are at 10.0% and 22.5% of the GPU-DG2 uniform grid at $\varepsilon = 10^{-3}$ and $\varepsilon = 10^{-4}$, respectively.

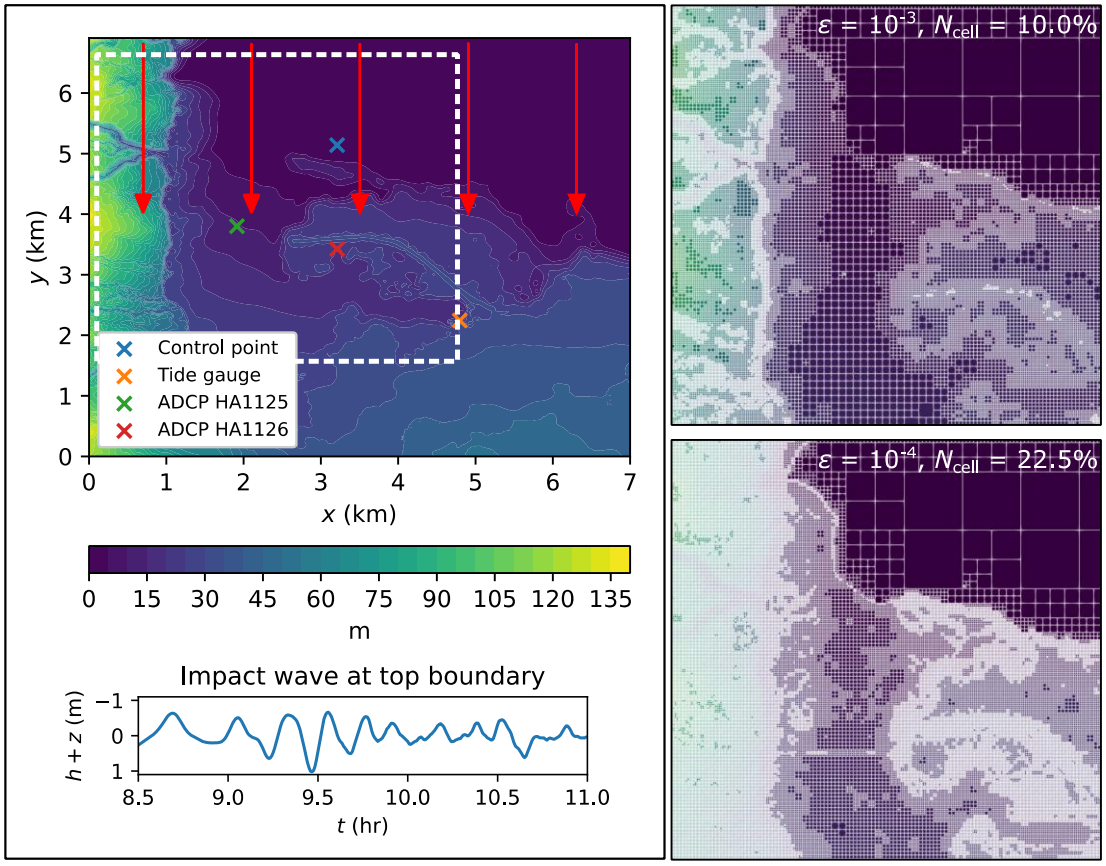

Figure 15: Hilo Harbour test case. Top-down view of bathymetry (top left panel), where the red arrows indicate the direction and distance travelled; tsunami time history entering the top boundary (bottom left panel); initial GPU-MWDG2 grids (right panels) for the potion in white box (top left panel).

In Figure 16, an analysis of runtimes of the GPU-MWDG2 and GPU-DG2 simulations is shown. Like the last test case (Sect. 3.3), the wave train enters the bathymetric area immediately and N_{cell} increases sharply and immediately to maximum values of 32% and 40% at $\varepsilon = 10^{-3}$ and 10^{-4}, respectively. The time history of N_{cell} stays at this maximum for the rest of the simulation except for localised drops at certain simulation times, e.g. at 8 hr. Following N_{cell}, R_{DG2} also increases sharply and immediately (to maximum values of 50% and 55% at $\varepsilon = 10^{-3}$ and 10^{-4}, respectively), and shows localised drops at the same times as the drops in N_{cell}. In contrast, the time history of R_{MRA} stays very flat throughout the simulation, except for a small, temporary drop at 8 hr, which is when the largest drop in N_{cell} occurs. Due to the generally flat time histories of R_{DG2} and R_{MRA}, the time history of S_{inst} is also flat except for localised peaks that occur at the same times as the localised drops in R_{DG2} and R_{MRA}. Unlike all of the previous test cases (Sects. 3.1 - 3.3), the time history of Δt is very similar between the GPU-DG2 simulation and the GPU-MWDG2 simulations, regardless of the ε value as they both yield high numbers of cells compared to the uniform GPU-DG2 grid, leading to highly refined non-uniform grids that are unlikely to have coarse cells compared to the previous test cases. Thus, the time history of $N_{\Delta t}$ is virtually identical across all simulations. Given that

the time histories of $N_{\Delta t}$, R_{DG2} and R_{MRA} are similar for both ε values, the time histories of C_{DG2} and C_{MRA} are also very similar, but importantly, due to the complex flows leading to highly refined non-uniform grids, the former dominates the latter in terms of the total computational effort. Overall, the time history of C_{tot} is very close between the GPU-DG2 simulation and the GPU-MWDG2 simulations in this test case (even more so than in the last case, Sect. 3.3), so S_{acc} is the lowest out of all the test cases, finishing at 1.25 and 1.10 using $\varepsilon = 10^{-3}$ and 10^{-4}, respectively.

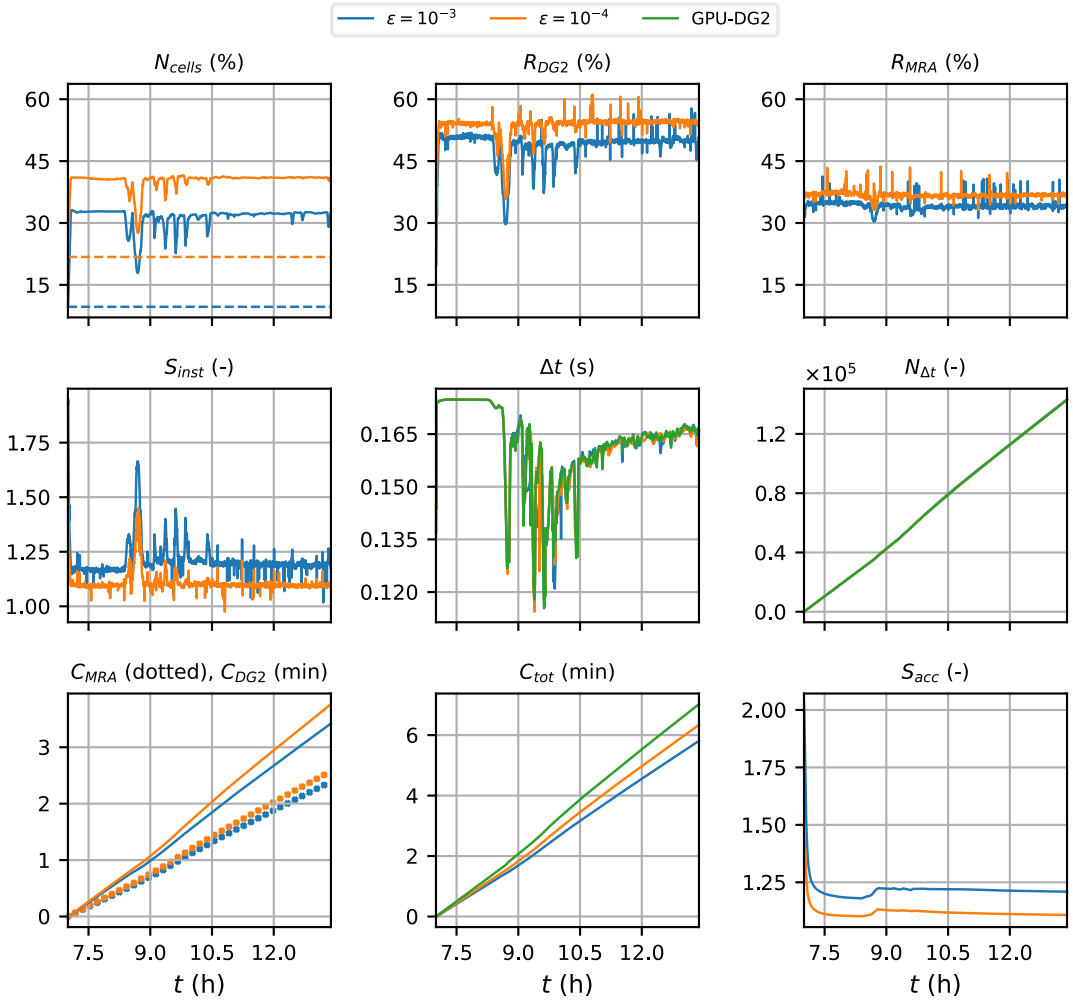

Figure 16: Hilo Harbour. Measuring the computational performance of the GPU-MWDG2 and GPU-DG2 simulations using the metrics of Table 1 to assess the speedup afforded by GPU-MWDG2 adaptivity: N_{cells},the number of cells in the GPU-MWDG2 non-uniform grid,where the dashed line represents the initial value at the beginning of the simulation; R_{DG2},the computational effort of performing the DG2 solver updates at given timestep relative to the GPU-DG2 simulation; R_{MRA}, the relative computational effort of performing the MRA process at a given timestep; S_{inst}, the instantaneous speedup achieved by the GPU-MWDG2 simulation over the GPU-DG2 simulation at a given timestep; Δt, the simulation timestep; $N_{\Delta t}$, the number of timesteps taken to reach a given simulation time; C_{DG2}, the cumulative computational effort of performing the DG2 solver updates up to a given simulation time; C_{MRA}, the cumulative computational effort of performing the MRA process; C_{tot}, the total cumulative computational effort; S_{acc}, the accumulated speedup of GPU-MWDG2 over GPU-DG2 up to a given simulation time.

Figure 17 shows the time series of the water surface elevation at observation points labelled "Control point" and "Tide gauge", and the time series of the u and v velocity components at the points labelled "ADCP HA1125" and "ADCP HA1126" (top left panel in Figure 15) predicted by GPU-DG2 using $\varepsilon = 10^{-3}$ and 10^{-4} and GPU-MWDG2. All the predictions are visually very similar to each other except for the u prediction, where some differences are seen for $\varepsilon = 10^{-3}$ and 10^{-4}. Compared to the measured data, both agreement and differences are seen, but it may not be possible to make a confident judgment due to the sparsity in the measured data—see the jagged shape of the black line.

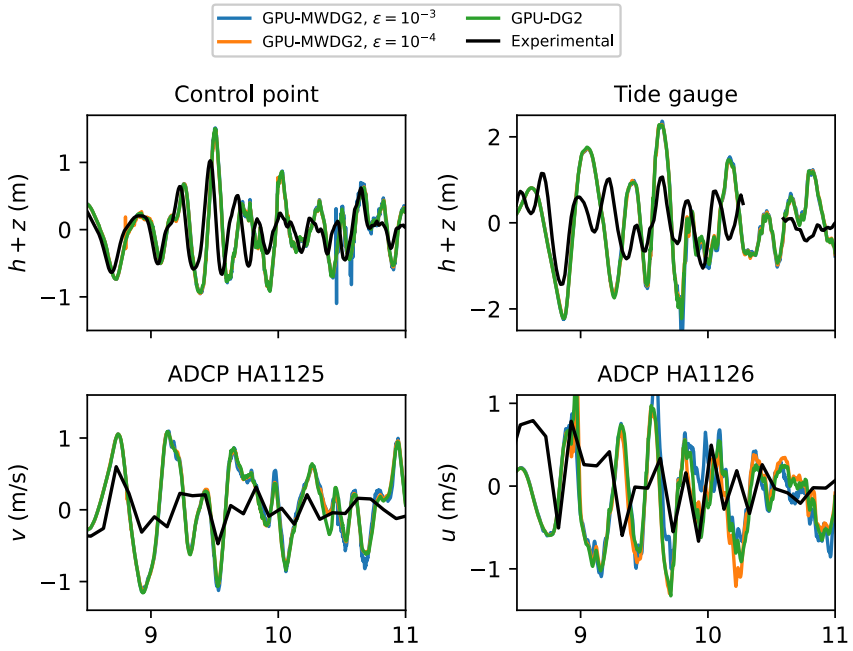

Figure 17: Hilo Harbour. Time series produced by GPU-DG2 and GPU-MWDG2, for the water surface elevation ($h + z$) at "control point" and "Tide gauge" (labelled in Figure 17), and for the velocity components at "ADCP HA1125" and ADCP HA 1126 (labelled in Figure 17), compared to the experimental results.

Table 6 shows the associated DE, FE and PE at the observation points. An outlier is noticeable in this test case for the FE at the Tide gauge and ADCP HA1125 observation points, which is actually smaller than the DE: this might be due missing data between 10 and 11 h (see the gap in the black line in the Tide gauge panel in Figure 17). Nonetheless, generally, the DE and FE in the same order of magnitude, while the PE reduces when ε is decreased from 10^{-3} to 10^{-4}, showing fulfilment of conditions (i) and (ii) and validating the choice of ε.

Table 6: Hilo Harbour. DE, FE and PE quantified using Eqs. (1) to (3) and considering measured vs predicted time series of the prognostic variable $h + z$ at observation points Tide gauge and Control point, and of the prognostic variables v and u at observation points ADCP 1125 and 1126 respectively. The errors are considered in order to check for fulfilment of conditions (i) and (ii) and confirm the validity of selecting $\varepsilon = 10^{-3}$ and 10^{-4} in this test case.

	DE	FE	PE

Observation point	**Quantity**	-	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$
Control point	$h + z$	0.380857	0.382853	0.381485	0.009817	0.007797
Tide gauge	$h + z$	0.936661	0.941419	0.931434	0.073245	0.051771
ADCP HA1125	v	0.514478	0.477909	0.50105	0.16383	0.221694
ADCP HA1126	u	0.609671	0.587934	0.598625	0.281472	0.263449

Overall, despite simulating a test case with a complex tsunami impact event and also a DEM size that requires selecting a small L = 10, GPU-MWDG2 still manages to attain speedups over GPU-DG2 in this test case: around 1.25 at $\varepsilon = 10^{-3}$ and 1.10 at $\varepsilon = 10^{-4}$. This seems to suggest that the GPU-MWDG2 solver can reliably be used to gain speedups over the GPU-DG2 solver even if simulating complex tsunami impact events, with both ε values being valid, using $\varepsilon = 10^{-3}$ to boost the speedup, or using $\varepsilon = 10^{-4}$ increase the quality of velocity predictions.

4 Conclusions and recommendations

This work reported the new LISFLOOD-FP 8.2 hydrodynamic modelling framework, which integrates the GPU parallelised grid resolution adaptivity of multiwavelets (MW) within the second-order discontinuous Galerkin (DG2) solver of the shallow water equations (GPU-MWDG2) to run simulations on a non-uniform grid. The GPU-MWDG2 solver enables dynamic (in time) grid resolution adaptivity based on both the (time-varying) flow solution and the (time-invariant) Digital Elevation Model (DEM) representations. It is aimed at reducing the runtime of the existing uniform grid GPU parallelised DG2 solver (GPU-DG2) for flood simulation driven by rapid, multiscale impact events, which were exemplified by tsunami-induced flooding.

The usability of the LISFLOOD-FP 8.2 hydrodynamic modelling framework was presented focusing on: how to run GPU-MWDG2 simulations from raster-formatted DEM and initial flow condition setup files from a user-specified maximum refinement level L (i.e. ratio of the DEM size to its resolution) and an error threshold ε (i.e. recommended to be between 10^{-4} and 10^{-3}); the effect of increasing L on GPU memory consumption and speedups; and, proposing new metrics for quantifying the speedups afforded by GPU-MWDG2 adaptivity against GPU-DG2 simulations.

The accuracy and efficiency of dynamic GPU-MWDG2 adaptivity was assessed for four laboratory- and field-scale tsunami-induced flood benchmarks featuring different impact event's complexity and duration (i.e. incorporating either single- or multi-peaked tsunamis and L = 10 or L = 12). GPU-MWDG2 simulation assessments were performed for $\varepsilon = 10^{-3}$ and 10^{-4}, which were also validated based on accuracy qualification with respect to benchmark-specific measured data. The accuracy assessment consistently confirms that an ε between 10^{-4} and 10^{-3} is valid: at $\varepsilon = 10^{-3}$ GPU-MWDG2 simulations

yield water level predictions as accurate as the GPU-DG2 predictions but using $\varepsilon = 10^{-4}$ can slightly improve velocity-related predictions.

Speedup assessments, based on instantaneous and cumulative metrics, suggested considerable gain when the DEM size and resolution involved $L \geq 10$ and for simulation durations that mostly spanned reduced-complexity events (i.e. single-peak tsunamis): As shown in Table 7, for single-peaked tsunamis, when the DEM required $L = 10$, GPU-MWDG2 was more than 2-fold faster than the GPU-DG2 simulations (i.e. "Monai Valley"), whereas with the DEM requiring $L = 12$, speedups of 3.3-to-4.5-fold could be achieved (i.e. "Seaside, Oregon"); in contrast, for multi-peaked tsunamis, GPU-MWDG2 speedups reduced to 1.8-fold for a DEM requiring $L = 12$ (i.e. "Tauranga Harbour") and to 1.2-fold for a smaller DEM needing $L = 10$ (i.e. "Hilo Harbour").

Table 7: Summary of GPU-MWDG2 runtimes and potential average speedups with respect to GPU-DG2.

				GPU-MWDG2				**GPU-DG2**
Tsunami (impact) event			***L* (*square uniform grid*)**	**Runtime**		**Speedup**		**Runtime**
				ε		ε		
Test case	t_{end}	Single-wave tsunami	----	10^{-3}	10^{-4}	10^{-3}	10^{-4}	----
Monai Valley	22.5 (9000 s*)	Yes	10 ($2^{10} \times 2^{10}$)	16 s	20 s	2.5	2.0	40 s
Seaside Oregon	40 s (33 min*)	Yes	12 ($2^{12} \times 2^{12}$)	3.5 min	5.2 min	4.5	3.3	13 min
Tauranga Harbour	40 hr	No (three peaks)	12 ($2^{12} \times 2^{12}$)	7.5 hr	8.1 hr	1.8	1.4	11.3 hr
Hilo Harbour	6 hr	No (eleven peaks)	10 ($2^{10} \times 2^{10}$)	5.3 min	5.8 min	1.3	1.2	6.9 min

*By accounting for the physical scaling factor of the replica.

In summary, the GPU-MWDG2 solver in LISFLOOD-FP 8.2 consistently accelerates GPU-DG2 simulations of rapid multiscale flooding flows, generally yielding the greatest speedups for simulations needing $L \geq 10$—due to its scalability on the GPU which results in larger speedup with increasing L—and driven by single-peaked impact events. Choosing ε closer to 10^{-3} would maximise speedup while choosing an ε closer 10^{-4} may be useful to particularly improve the velocity-related predictions. The LISFLOOD-FP 8.2 code and the simulated benchmarks' set-up files and datasets are available under the code and data availability statement, while a video tutorial is available under the video supplement statement; further documentation is available at https://www.seamlesswave.com/Adaptive (last accessed: 7 July 2025).

Code and data availability. LISFLOOD-FP 8.2 source code is available from Zenodo (LISFLOOD-FP developers, 2024; https://zenodo.org/doi/10.5281/zenodo.4073010) as well as the simulation results and input files/scripts for reproducing them (Chowdhury, 2024; https://doi/org/10.5281/zenodo.13909072); the GPU-MWDG2 solver code is also available at https://github.com/al0vya/gpu-mwdg2.

Video supplement. Step-by-step video tutorial on how to download, install and run the GPU-MWDG2 code for the Monai Valley" example (Sect. 3.1) is available from Zenodo (Chowdhury, 2025; https://zenodo.org/records/15851523). The video demo also includes updates on the changes made to the CMake build process for compatibility with different versions of the CUDA toolkit.

Author contributions. AAC coded, optimised and integrated the GPU-MWDG2 into the LISFLOOD-FP framework (methodology; software; validation; investigation; data curation; visualization; formal analysis). GK contributed to the conceptualisation, formal analysis, funding acquisition and project administration. Both AAC and GK conceived and wrote the paper (original draft; review and editing).

Competing interests. The contact author has declared that none of the authors has any competing interests.

Acknowledgements. The authors are extremely grateful for Charles Rougé from the University of Sheffield for his feedback on the efficiency analysis of Sect. 3, and Paul Bates and Jefferey Neal from the University of Bristol for their support of the collaborative open source LISFLOOD-FP project.

Financial Support. AAC and GK were supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/R007349/1.

References

Arcos, M., & LeVeque, R. (2014). Validating Velocities in the GeoClaw Tsunami Model using Observations Near Hawaii from the 2011 Tohoku Tsunami. *Pure and Applied Geophysics*, *172*. https://doi.org/10.1007/s00024-014-0980-y

Ayog, J. L., Kesserwani, G., & Baú, D. (2021). Well-resolved velocity fields using discontinuous Galerkin shallow water solutions. In *arXiv [physics.flu-dyn]*. http://arxiv.org/abs/2104.11308

Bates, P. D., Horritt, M. S., & Fewtrell, T. J. (2010). A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling. *Journal of Hydrology*, *387*(1), 33–45. https://doi.org/https://doi.org/10.1016/j.jhydrol.2010.03.027

Berger, M. J., George, D. L., LeVeque, R. J., & Mandli, K. T. (2011). The GeoClaw software for depth-averaged flows with adaptive refinement. *Advances in Water Resources*, *34*(9), 1195–1206. https://doi.org/10.1016/J.ADVWATRES.2011.02.016

Blaise, S., & St-Cyr, A. (2012). A Dynamic hp-Adaptive Discontinuous Galerkin Method for Shallow-Water Flows on the Sphere with Application to a Global Tsunami Simulation. *Monthly Weather Review*, *140*(3), 978–996. https://doi.org/https://doi.org/10.1175/MWR-D-11-00038.1

Blaise, S., St-Cyr, A., Mavriplis, D., & Lockwood, B. (2013). Discontinuous Galerkin unsteady discrete adjoint method for real-time efficient tsunami simulations. *Journal of Computational Physics*, *232*(1), 416–430. https://doi.org/https://doi.org/10.1016/j.jcp.2012.08.022

Bonev, B., Hesthaven, J. S., Giraldo, F. X., & Kopera, M. A. (2018). Discontinuous Galerkin scheme for the spherical shallow water equations with applications to tsunami modeling and prediction. *Journal of Computational Physics*, *362*, 425–448. https://doi.org/https://doi.org/10.1016/j.jcp.2018.02.008

Borrero, J. C., LeVeque, R. J., Greer, S. D., O'Neill, S., & Davis, B. N. (2015). Observations and modelling of tsunami currents at the port of Tauranga, New Zealand. *Australasian Coasts & Ports Conference 2015: 22nd Australasian Coastal and Ocean Engineering Conference and the 15th Australasian Port and Harbour Conference*. https://search.informit.org/doi/10.3316/informit.703156566786424

Castro, C. E., Behrens, J., & Pelties, C. (2016). Optimization of the ADER-DG method in GPU applied to linear hyperbolic PDEs. *International Journal for Numerical Methods in Fluids*, *81*(4), 195–219. https://doi.org/https://doi.org/10.1002/fld.4179

Caviedes-Voullième, D., Gerhard, N., Sikstel, A., & Müller, S. (2020). Multiwavelet-based mesh adaptivity with Discontinuous Galerkin schemes: Exploring 2D shallow water problems. *Advances in Water Resources*, *138*, 103559. https://doi.org/10.1016/J.ADVWATRES.2020.103559

Caviedes-Voullième, D., & Kesserwani, G. (2015). Benchmarking a multiresolution discontinuous Galerkin shallow water model: Implications for computational hydraulics. *Advances in Water Resources*, *86*, 14–31. https://doi.org/10.1016/J.ADVWATRES.2015.09.016

Chowdhury, A. (2025). *Simulation run guide for the GPU-MWDG2 solver in LISFLOOD-FP 8.2*. https://zenodo.org/records/15851523

Chowdhury, A. A., & Kesserwani, G. (2024). *Dataset for the paper "LISFLOOD-FP 8.2: Dynamic multiwavelet grid resolution adaptivity for faster, GPU-accelerated discontinuous Galerkin simulations of rapid multiscale floods."* https://doi.org/https://zenodo.org/records/10779500

Chowdhury, A. A., Kesserwani, G., Rougé, C., & Richmond, P. (2023). GPU-parallelisation of Haar wavelet-based grid resolution adaptation for fast finite volume modelling: application to shallow water flows. *Journal of Hydroinformatics*, *25*(4), 1210–1234. https://doi.org/10.2166/hydro.2023.154

de la Asunción, M., & Castro, M. J. (2017). Simulation of tsunamis generated by landslides using adaptive mesh refinement on GPU. *Journal of Computational Physics*, *345*, 91–110. https://doi.org/10.1016/J.JCP.2017.05.016

Ferreira, C. R., & Bader, M. (2017). Load Balancing and Patch-Based Parallel Adaptive Mesh Refinement for Tsunami Simulation on Heterogeneous Platforms Using Xeon Phi Coprocessors. *Proceedings of the Platform for Advanced Scientific Computing Conference*, 12. https://doi.org/10.1145/3093172.3093237

Gao, S., Collecutt, G., Syme, W. J., & Ryan, P. (2020). *HIGH RESOLUTION NUMERICAL MODELLING OF TSUNAMI INUNDATION USING QUADTREE METHOD AND GPU ACCELERATION*.

Gerhard, N., Caviedes-Voullième, D., Müller, S., & Kesserwani, G. (2015). Multiwavelet-based grid adaptation with discontinuous Galerkin schemes for shallow water equations. *Journal of Computational Physics*, *301*, 265–288. https://doi.org/10.1016/J.JCP.2015.08.030

Gerhard, N., & Müller, S. (2016). Adaptive multiresolution discontinuous Galerkin schemes for conservation laws: multi-dimensional case. *Computational and Applied Mathematics*, *35*(2), 321–349. https://doi.org/10.1007/s40314-014-0134-y

Hajihassanpour, M., Bonev, B., & Hesthaven, J. S. (2019). A comparative study of earthquake source models in high-order accurate tsunami simulations. *Ocean Modelling*, *141*, 101429. https://doi.org/https://doi.org/10.1016/j.ocemod.2019.101429

Hunter, N. M., Horritt, M. S., Bates, P. D., Wilson, M. D., & Werner, M. G. F. (2005). An adaptive time step solution for raster-based storage cell modelling of floodplain inundation. *Advances in Water Resources*, *28*(9), 975–991. https://doi.org/https://doi.org/10.1016/j.advwatres.2005.03.007

Kesserwani, G. (2013). Topography discretization techniques for Godunov-type shallow water numerical models: a comparative study. *Journal of Hydraulic Research*, *51*(4), 351–367. https://doi.org/10.1080/00221686.2013.796574

Kesserwani, G., Ayog, J. L., & Bau, D. (2018). Discontinuous Galerkin formulation for 2D hydrodynamic modelling: Trade-offs between theoretical complexity and practical convenience. *Computer Methods in Applied Mechanics and Engineering*, *342*, 710–741. https://doi.org/10.1016/J.CMA.2018.08.003

Kesserwani, G., Caviedes-Voullième, D., Gerhard, N., & Müller, S. (2015). Multiwavelet discontinuous Galerkin h-adaptive shallow water model. *Computer Methods in Applied Mechanics and Engineering*, *294*, 56–71. https://doi.org/https://doi.org/10.1016/j.cma.2015.05.016

Kesserwani, G., & Liang, Q. (2012). Dynamically adaptive grid based discontinuous Galerkin shallow water model. *Advances in Water Resources*, *37*, 23–39. https://doi.org/10.1016/J.ADVWATRES.2011.11.006

Kesserwani, G., & Sharifian, M. K. (2020). (Multi)wavelets increase both accuracy and efficiency of standard Godunov-type hydrodynamic models: Robust 2D approaches. *Advances in Water Resources*, *144*, 103693. https://doi.org/10.1016/J.ADVWATRES.2020.103693

Kesserwani, G., & Sharifian, M. K. (2023). (Multi)wavelet-based Godunov-type simulators of flood inundation: Static versus dynamic adaptivity. *Advances in Water Resources*, *171*, 104357. https://doi.org/https://doi.org/10.1016/j.advwatres.2022.104357

Kesserwani, G., Shaw, J., Sharifian, M. K., Bau, D., Keylock, C. J., Bates, P. D., & Ryan, J. K. (2019). (Multi)wavelets increase both accuracy and efficiency of standard Godunov-type hydrodynamic models. *Advances in Water Resources*, *129*, 31–55. https://doi.org/10.1016/J.ADVWATRES.2019.04.019

Kesserwani, G., & Wang, Y. (2014). Discontinuous Galerkin flood model formulation: Luxury or necessity? *Water Resources Research*, *50*(8), 6522–6541. https://doi.org/https://doi.org/10.1002/2013WR014906

Kevlahan, N. K.-R., & Lemarié, F. (2022). wavetrisk-2.1: an adaptive dynamical core for ocean modelling. *Geosci. Model Dev.*, *15*(17), 6521–6539. https://doi.org/10.5194/gmd-15-6521-2022

Lee, H. S. (2016). *Tsunami Run-up Modeling with Adaptive Mesh Refinement Method: A Case Study for Monai Village Run-up Experiment*.

LeVeque, R. J., George, D. L., & Berger, M. J. (2011). Tsunami modelling with adaptively refined finite volume methods. *Acta Numerica*, *20*, 211–289. https://doi.org/DOI: 10.1017/S0962492911000043

Liang, Q., Hou, J., & Amouzgar, R. (2015). Simulation of Tsunami Propagation Using Adaptive Cartesian Grids. *Coastal Engineering Journal*, *57*(4), 1550016-1-1550016–1550030. https://doi.org/10.1142/S0578563415500163

Lynett, P. J., Gately, K., Wilson, R., Montoya, L., Arcas, D., Aytore, B., Bai, Y., Bricker, J. D., Castro, M. J., Cheung, K. F., David, C. G., Dogan, G. G., Escalante, C., González-Vida, J. M., Grilli, S. T., Heitmann, T. W., Horrillo, J., Kânoğlu, U., Kian, R., … Zhang, Y. J. (2017). Inter-model analysis of tsunami-induced coastal currents. *Ocean Modelling*, *114*, 14–32. https://doi.org/https://doi.org/10.1016/j.ocemod.2017.04.003

Macías, J., Castro, M. J., & Escalante, C. (2020). Performance assessment of the Tsunami-HySEA model for NTHMP tsunami currents benchmarking. Laboratory data. *Coastal Engineering*, *158*, 103667. https://doi.org/10.1016/J.COASTALENG.2020.103667

Macías, J., Castro, M. J., Ortega, S., & González-Vida, J. M. (2020). Performance assessment of Tsunami-HySEA model for NTHMP tsunami currents benchmarking. Field cases. *Ocean Modelling*, *152*, 101645. https://doi.org/https://doi.org/10.1016/j.ocemod.2020.101645

Macías, J., Castro, M., Ortega, S., Escalante Sánchez, C., & González Vida, J. (2015). Tsunami currents benchmarking results for Tsunami-HySEA. In *Report on the 2015 NTHMP Current Modeling Workshop*. https://doi.org/10.13140/RG.2.2.22999.47527

Matsuyama, M., & Tanaka, H. (2001). *An experimental study of the highest run-up height in the 1993 Hokkaido Nansei-Oki earthquake tsunami*. *7*.

Nandi, S., & Reddy, M. J. (2022). An integrated approach to streamflow estimation and flood inundation mapping using VIC, RAPID and LISFLOOD-FP. *Journal of Hydrology*, *610*, 127842. https://doi.org/https://doi.org/10.1016/j.jhydrol.2022.127842

NVIDIA. (2023). *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/

Park, H., Cox, D. T., Lynett, P. J., Wiebe, D. M., & Shin, S. (2013). Tsunami inundation modeling in constructed environments: A physical and numerical comparison of free-surface elevation, velocity, and momentum flux. *Coastal Engineering*, *79*, 9–21. https://doi.org/https://doi.org/10.1016/j.coastaleng.2013.04.002

Park, J., Yuk, J.-H., Joo, W., & Lee, H. S. (2019). Wave Run-up Modeling with Adaptive Mesh Refinement (AMR) Method in the Busan Marine City during Typhoon Chaba (1618). *Journal of Coastal Research*, *91*, 56. https://doi.org/10.2112/SI91-012.1

Popinet, S. (2011). Quadtree-adaptive tsunami modelling. *Ocean Dynamics*, *61*(9), 1261–1285. https://doi.org/10.1007/s10236-011-0438-z

Popinet, S. (2012). Adaptive modelling of long-distance wave propagation and fine-scale flooding during the Tohoku tsunami. *Nat. Hazards Earth Syst. Sci.*, *12*(4), 1213–1227. https://doi.org/10.5194/nhess-12-1213-2012

Popinet, S., & Rickard, G. (2007). A tree-based solver for adaptive ocean modelling. *Ocean Modelling*, *16*(3), 224–249. https://doi.org/https://doi.org/10.1016/j.ocemod.2006.10.002

Qin, X., Motley, M., LeVeque, R., Gonzalez, F., & Mueller, K. (2018). A comparison of a two-dimensional depth-averaged flow model and a three-dimensional RANS model for predicting tsunami inundation and fluid forces. *Nat. Hazards Earth Syst. Sci.*, *18*(9), 2489–2506. https://doi.org/10.5194/nhess-18-2489-2018

Rannabauer, L., Dumbser, M., & Bader, M. (2018). ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework. *Computers & Fluids*, *173*, 299–306. https://doi.org/https://doi.org/10.1016/j.compfluid.2018.01.031

Sharifian, M. K., Hassanzadeh, Y., Kesserwani, G., & Shaw, J. (2019). Performance study of the multiwavelet discontinuous Galerkin approach for solving the Green-Naghdi equations. *International Journal for Numerical Methods in Fluids*, *90*(10), 501–521. https://doi.org/https://doi.org/10.1002/fld.4732

Sharifian, M. K., Kesserwani, G., Chowdhury, A. A., Neal, J., & Bates, P. (2023). LISFLOOD-FP 8.1: new GPU-accelerated solvers for faster fluvial/pluvial flood simulations. *Geosci. Model Dev.*, *16*(9), 2391–2413. https://doi.org/10.5194/gmd-16-2391-2023

Sharifian, M. K., Kesserwani, G., & Hassanzadeh, Y. (2018). A discontinuous Galerkin approach for conservative modeling of fully nonlinear and weakly dispersive wave transformations. *Ocean Modelling*, *125*, 61–79. https://doi.org/https://doi.org/10.1016/j.ocemod.2018.03.006

Shaw, J., Kesserwani, G., Neal, J., Bates, P., & Sharifian, M. K. (2021). LISFLOOD-FP 8.0: the new discontinuous Galerkin shallow-water solver for multi-core CPUs and GPUs. *Geosci. Model Dev.*, *14*(6), 3577–3602. https://doi.org/10.5194/gmd-14-3577-2021

Sun, X., Kesserwani, G., Sharifian, M. K., & Stovin, V. (2023). Simulation of laminar to transitional wakes past cylinders with a discontinuous Galerkin inviscid shallow water model. *Journal of Hydraulic Research*, *61*(5), 631–650. https://doi.org/10.1080/00221686.2023.2239750

Velioglu Sogut, D., & Yalciner, A. C. (2019). Performance Comparison of NAMI DANCE and FLOW-3D® Models in Tsunami Propagation, Inundation and Currents using NTHMP Benchmark Problems. *Pure and Applied Geophysics*, *176*(7), 3115–3153. https://doi.org/10.1007/s00024-018-1907-9

Violeau, D., Ata, R., Benoit, M., Joly, A., Abadie, S., Clous, L., Martin Medina, M., Morichon, D., Chicheportiche, J., Le Gal, M., Gailler, A., Hebert, H., Imbert, D., Kazolea, M., Ricchiuto, M., Le Roy, S., Pedreros, R., Rousseau, M., Pons, K., … Silva Jacinto, R. (2016). *Database of Validation Cases for Tsunami Numerical Modelling*.

Zeng, Z., Wang, Z., & Lai, C. (2022). Simulation Performance Evaluation and Uncertainty Analysis on a Coupled Inundation Model Combining SWMM and WCA2D. *International Journal of Disaster Risk Science*, *13*(3), 448–464. https://doi.org/10.1007/s13753-022-00416-3

Ziliani, L., Surian, N., Botter, G., & Mao, L. (2020). Assessment of the geomorphic effectiveness of controlled floods in a braided river using a reduced-complexity numerical model. *Hydrol. Earth Syst. Sci.*, *24*(6), 3229–3250. https://doi.org/10.5194/hess-24-3229-2020

Appendix A: The GPU-MWDG2 algorithm

The GPU-MWDG2 algorithm that is integrated into LISFLOOD-FP 8.2 solves the two-dimensional shallow water equations over a non-uniform grid that locally adapts its grid resolution to the flow solutions and DEM representation every simulation timestep, Δt. The conservative form of the shallow water equations in vectorial format is as follows:

$$\partial_t \boldsymbol{U} + \partial_x \boldsymbol{F}(\boldsymbol{U}) + \partial_y \boldsymbol{G}(\boldsymbol{U}) = \boldsymbol{S}_b(\boldsymbol{U}) + \boldsymbol{S}_f(\boldsymbol{U}) \quad \text{(A1)}$$

Where ∂ represents a partial derivative operator; $\boldsymbol{U} = [h, hu, hv]^T$ is the vector of the flow variables where T stands for the transpose operator, $h(x, y, t)$ is the water depth (m) at time t and location (x, y), and $u(x, y, t)$ and $v(x, y, t)$ are the x- and y-component of the velocity field (m/s) in two-dimensional Cartesian space; $\boldsymbol{F} = [hu, (hu)^2 h^{-1} + 0.5gh^2, huv]^T$ and $\boldsymbol{G} = [hv, huv, (hv)^2 h^{-1} + 0.5gh^2]^T$ are the components of the flux vector in which g is the gravitational acceleration constant (m/s^2); $\boldsymbol{S}_b = [0, -gh\partial_x z, -gh\partial_y z]^T$ is the bed-slope source term vector incorporating the partial derivative of the bed elevation function $z(x, y)$; and $\boldsymbol{S}_f = [0, -C_f u\sqrt{u^2 + v^2}, -C_f v\sqrt{u^2 + v^2}]^T$ is the friction source term vector including the friction effects as function of $C_f = gn_M^2 h^{-1/3}$ in which n_M is Manning's roughness parameter. For ease of presentation, the scalar variable s will hereafter be used to represent any of the physical flow quantities in $\boldsymbol{U}$ as well as the bed elevation z.

Over each computational cell c, the DG2 modelled data for any of the any physical flow quantities, $s \in \{h, hu, hv\}$, follows a piecewise-planar solution, denoted by $s_c(x, y, t)$ (Kesserwani & Sharifian, 2020). The piecewise-planar solution, $s_c(x, y, t)$, is expanded onto local basis functions from the scaled and truncated Legendre basis (Kesserwani et al., 2018; Kesserwani & Sharifian, 2020) to become spanned by three shape coefficients: $s_c = [s_c^0(t), s_c^{1x}(t), s_c^{1y}(t)]^T$, where $s_c^0(t)$ is an average coefficient; and $s_c^{1x}(t)$ and $s_c^{1y}(t)$ are x- and y-directional slope coefficients, respectively (see Eq. 10 in

Kesserwani & Sharifian, 2020). The bed elevation, $s \in \{z\}$, is also represented as piecewise-planar, but it is spanned by time-independent shape coefficients. The shape coefficients in s_c, i.e. $s \in \{h, hu, hv, z\}$, must be initialised (Eq. 11 in Kesserwani & Sharifian, 2020), while only the time-dependent ones, i.e. $s \in \{h, hu, hv\}$, are updated using "DG2 solver updates" by an explicit two-stage Runge-Rutta scheme solving three ordinary differential equations:

$$\partial_t s_{c\square}(t) = L_c \tag{A2}$$

Where $L_c = [L_c^0, L_c^{1x}, L_c^{1y}]^T$ includes the respective components of the local discrete spatial DG2 operators to update each of the coefficients in $s_{c\square} = [s_c^0(t), s_c^{1x}(t), s_c^{1y}(t)]^T$. The operators in L_c were already designed to preserve mimetic properties by incorporating robust treatments of the bed and friction source terms and of moving wet-dry fronts (Kesserwani & Sharifian, 2020; Shaw et al., 2021).

The MWDG2 algorithm involves the MRA procedure to decompose, analyse and assemble the shape coefficients $s_{c\square}$, $s \in \{h, hu, hv, z\}$ into a non-uniform grid over which the DG2 solver updates are applied (Eq. A2). The MWDG2 algorithm was substantially redesigned to enable efficient parallelisation on the GPU (Chowdhury et al., 2023; Kesserwani & Sharifian, 2023): next, an overview of the GPU parallelised MWDG2 algorithm (GPU-MWDG2) is provided.

In the CUDA programming model for parallelisation the GPU (NVIDIA, 2023), instructions are executed in parallel by workers called "threads", and a group of 32 consecutive threads that operate in lockstep is called a "warp". To devise an efficiently parallelised GPU-MWDG2 code, coalesced memory access, occurring when threads in a warp access contiguous memory locations, should be maximised, and warp divergence, occurring when threads within a single warp perform different instructions, should be minimised. To achieve these requirements in the GPU-MWDG2 code, the implementation of the MRA procedure had to be reformulated so as to ensure the DG2 solver updates are applicable cell-wise in a coalesced manner just like the GPU-parallelised DG2 solver (GPU-DG2) that runs on a uniform grid (Shaw et al., 2021).

A.1. MRA procedure

The MRA procedure must start from a *square uniform grid* at the finest resolution, in particular at the given DEM resolution, which is defined to have a maximum refinement level, L. This finest grid contains $2^L \times 2^L$ cells, on which the shape coefficients $s_c^{(L)}$ are initialised. From the finest grid, the MRA procedure can be applied to build a hierarchy of grids of successively coarser resolution at levels $n = L - 1, \ldots, 1, 0$, comprising $2^n \times 2^n$ cells each. Using the "encoding" operation, the shape coefficients, $s_c^{(n)}$, and their associated "details", $d_{c,\Theta}^{(n)} = [d_{c,\Theta}^{0,(n)}, d_{c,\Theta}^{1x,(n)}, d_{c,\Theta}^{1y,(n)}]^T$, $\Theta = \alpha, \beta, \gamma$, can be produced on the "parent" cells of the coarser resolution grids, at level n, from the shape coefficients $s_{[0]}^{(n+1)}$, $s_{[1]}^{(n+1)}$, $s_{[2]}^{(n+1)}$ and $s_{[3]}^{(n+1)}$ of the four "children" cells at the finer resolution grids, at level $n + 1$ (Eq. 30 in Kesserwani & Sharifian, 2020). With the GPU-MWDG2 solver, $s_c^{(n)}$ and $d_{c,\Theta}^{(n)}$ are stored in arrays in GPU memory that are indexed using Z-order curves (Chowdhury et al., 2023), as exemplified in Figure A1a for a simplistic case with $L = 2$. Hence, $s_{[0]}^{(n+1)}, s_{[1]}^{(n+1)}, s_{[2]}^{(n+1)}$ and $s_{[3]}^{(n+1)}$ all reside in adjacent memory locations when used to produce $s_c^{(n)}$ and $d_{c,\Theta}^{(n)}$ (see Figure A1b) and can be accessed using vectorised

`float4` instructions (see lines 7 to 22 of Algorithm 1) to ensure coalesced memory access. The produced $s_c^{(n)}$ can then be trivially stored in the appropriate index of the array (see lines 25 to 27 of Algorithm 1), again in a coalesced manner due to the self-similar nature of the Z-order curve between refinement levels n and $n + 1$.

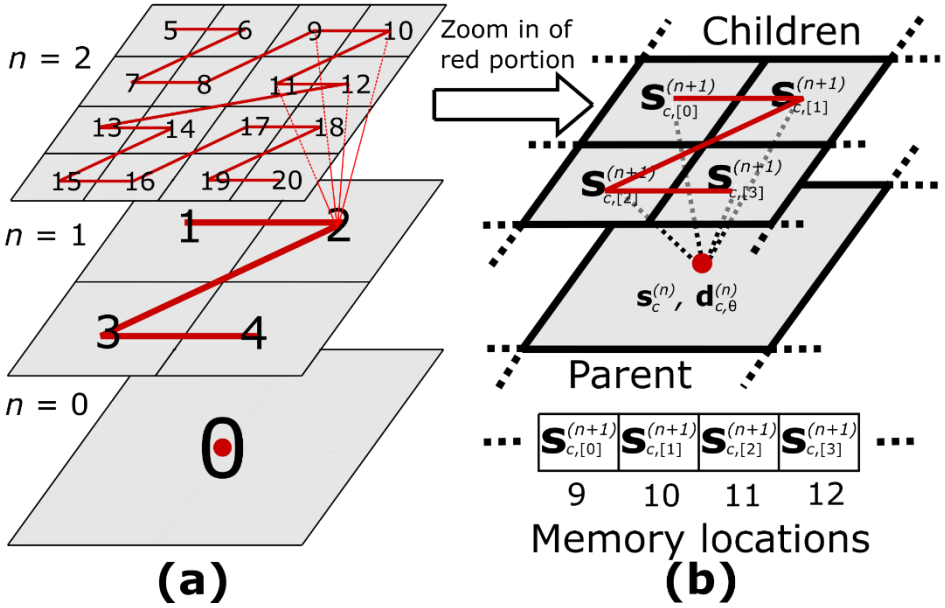

Figure A1: Indexing and storage for the MRA procedure on the GPU. Left panel shows a hierarchy of grids across which cells are indexed along the Z-order curve. Right panel shows how four "children" cells at resolution level $n + 1$, and their "parent" cell at resolution level n, noting that the shape coefficients at the "children" cells are stored in adjacent GPU memory locations.

```
01 __global__
02 void encode_flow_kernel_mw
03 (
04     // Inputs
05 )
06 {
07     // Compute Z order indices based on thread index
08     HierarchyIndex idx = blockIdx.x * blockDim.x + threadIdx.x;
09     HierarchyIndex curr_lvl_idx = get_lvl_idx(level);
10     HierarchyIndex next_lvl_idx = get_lvl_idx(level + 1);
11     HierarchyIndex parent_idx = curr_lvl_idx + idx;
12     HierarchyIndex child_idx = next_lvl_idx + 4 * (parent_idx - curr_lvl_idx);
13
14     // Load 4 child scale coefficients in a coalesced manner using vector loads (float4)
15     load_children_vector
16     (
17         children,
18         d_scale_coeffs.eta0,
19         d_scale_coeffs.eta1x,
20         d_scale_coeffs.eta1y,
21         child_idx
22     );
23
24     // Use encoding equation to compute and store parent scale coefficient
25     d_scale_coeffs.eta0[parent_idx]  = encode_scale_0 (children);
26     d_scale_coeffs.eta1x[parent_idx] = encode_scale_1x(children);
27     d_scale_coeffs.eta1y[parent_idx] = encode_scale_1y(children);
28 }
```

Algorithm 1: Optimised CUDA kernel for performing the encoding operation in parallel. Coalesced memory access is ensured because the Z-order indices computed per thread point to adjacent memory locations in the array that is used to $s_c^{(n)}$ and $d_{c,\Theta}^{(n)}$. As seen in line 15, vectorised float4 loads can be used to load the children when computing the parent. Storing the parent is trivially coalesced due to the self-similar nature of Z-order indexing across refinement levels n and $n + 1$.

While encoding, the magnitude of all the details $d_{c,\Theta}^{(n)}$ is analysed against epsilon in order to identify significant details (Kesserwani & Sharifian, 2020), which results in a tree-like structure of significant details (Figure A2a). The MRA procedure then uses this tree to generate the non-uniform grid via the "decoding" operation. Decoding is applied within the hierarchy of grids, starting from the coarsest resolution grid until reaching a" leaf" cell where significant details reside (i.e.

where a branch of the tree terminates, see Figure A2a where leaf cells are coloured). After decoding, the shape vectors $s_{[0]}^{(n+1)}$, $s_{[1]}^{(n+1)}$, $s_{[2]}^{(n+1)}$ and $s_{[3]}^{(n+1)}$ at the leaf cells are stored on the non-uniform grid (Eq. 31 in Kesserwani & Sharifian, 2020) to be updated in time.

With the GPU-MWDG2 solver, decoding must be performed using a parallel tree traversal algorithm (PTT) to minimise warp divergence (Chowdhury et al., 2023). To do so, the PTT starts by launching as many threads t_n as the number of cells on the finest resolution grid; for example, 16 threads t_0 to t_{15} for traversing the tree in Figure A2a. Each thread independently traverses the tree starting from the cell on the coarsest resolution grid until it reaches its leaf cell c, i.e. until it reaches either a detail that is no longer significant (line 14 of Algorithm 2) or the finest refinement level (line 28 of Algorithm 2). Once a thread reaches a leaf cell, it records the Z-order index of that leaf cell (Figures A2b and A2c).

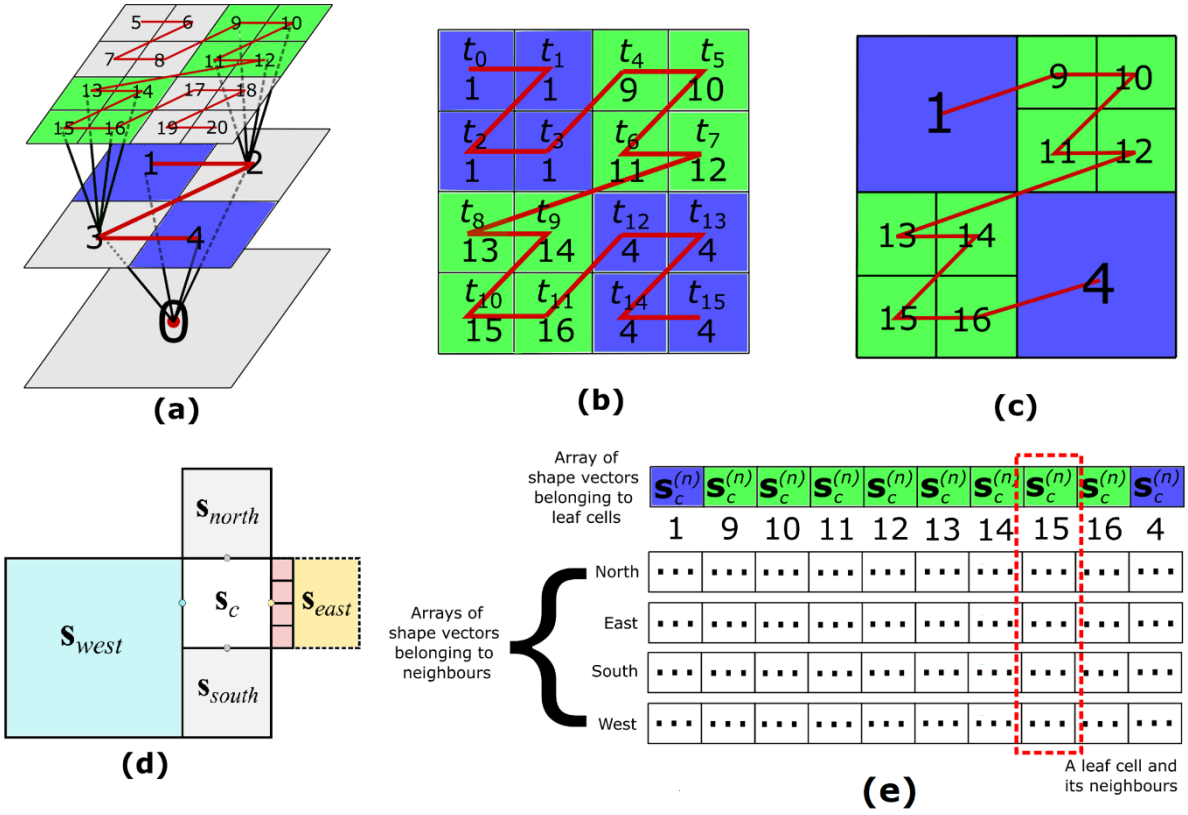

Figure A2: Parallel tree traversal (PTT) and neighbour finding. (a) The tree-like structure obtained after flagging significant details during the process of encoding; (b) The leaf cells where the tree terminates (highlighted in green and blue); (c) Leaf cells are assembly into the non-uniform grid; (e) Possible scenarios of neighbouring cells to leaf cell c; and (e) leaf and neighbour cells storage as arrays in GPU memory.

```
__global__
void traverse_tree_of_sig_details
(
    // Inputs
)
{
    while (keep_on_traversing)
    {
        MortonCode curr_code = ( fine_code >> ( 2 * (solver_params.L - level) ) );
        HierarchyIndex curr_lvl_idx = get_lvl_idx(level);
        HierarchyIndex h_idx = curr_lvl_idx + curr_code;
        bool is_sig = d_sig_details[h_idx];

        if (!is_sig)
        {
            // Record Z order index
            keep_on_traversing = false;
        }
        else
        {
            if (!penultimate_level)
            {
                keep_on_traversing = true;
            }
            else
            {
                // Record Z order index
                keep_on_traversing = false;
            }
        }
    }
}
```

Algorithm 2: Optimised CUDA kernel for performing parallel tree traversal (PTT) that minimises warp divergence. Threads keep traversing the tree of significant details until they reach either an insignificant detail or the finest refinement level. Warp divergence is reduced because threads that end up reaching nearby leaf cells have similar traversal paths.

Figure A2b shows the indices of the leaf cells identified by each thread once PTT is complete. Since the PTT started with 16 threads and there are fewer leaf cells than the threads, many of threads ended up identifying the same index of the leaf cell (e.g., t_0 to t_3 identified the leaf cell with index 1 and t_{12} to t_{15} identified the leaf cell with index 4, all having finished execution at line 17 of Algorithm 2). Threads with duplicate indices are re-used, alongside the other threads, to search and record the indices of east, west, north, and south neighbouring cells of each leaf cell by making each thread look up, down, left, and right. For example, t_0 to t_3 of the leaf cell with index 1 will identify the east neighbour cells 9 and 11 (Figure A2c). Since the DG2 solver updates on the leaf cell with index 1 requires the (shape coefficients of the) east neighbour cells (shaded red, Figure A2d) to be at the same resolution level as the coarser leaf cell, the PPT will instead record the index of coarsened east neighbour cells (yellow shaded, Figure A2d). For any other scenario (e.g., west, north, or south neighbour cells in Figure A2d), the azctual indices and shape coefficients are recorded by the PTT. After recording the indices and shape coefficients, $s_c^{(n)}$, which are to unique each leaf cell c, and its neighbours, $s_{north}^{(n)}$, $s_{south}^{(n)}$, $s_{east}^{(n)}$ and $s_{west}^{(n)}$, they are stored on the non-uniform grid (Figure A2c). In particular, shape coefficients for the leaf cells, $s_c^{(n)}$, are stored in a separate, contiguous arrays in GPU memory, and separate, contiguous arrays are also used to do the same for the shape coefficients of their neighbour cells, $s_{north}^{(n)}$, $s_{south}^{(n)}$, $s_{east}^{(n)}$ and $s_{west}^{(n)}$ (see Figure A2e). With this cell-wise storage of indices and shape coefficients in contiguous arrays, the DG2 solver updates, Eq. A2, can be applied in a coalesced manner.

A.2 DG2 solver update on the non-uniform grid

On the non-uniform grid, the DG2 solver updates, Eq. A2, are applied to update the shape coefficients $s_c^{(n)}$ by half a timestep over the first Runge-Kutta time stage. After this, another re-encoding step must be applied to the update shape coefficients $s_c^{(n)}$ so that the stored shape coefficients of the four neighbours, $s_{north}^{(n)}$, $s_{south}^{(n)}$, $s_{east}^{(n)}$ and $s_{west}^{(n)}$, are also lifted by half a timestep. Now, the shape coefficients $s_c^{(n)}$ can be updated by a full timestep by completing the second of Runge-Kutta time stage.

Appendix B: Step-by-step instructions for running the "Monai valley" example

This Appendix shows how to run a simulation of the "Monai valley" example (Sect. 3.1) using the GPU-MWDG2 solver step-by-step. To use the GPU-MWDG2 solver, the LISFLOOD-FP source code has to be downloaded (LISFLOOD-FP developers, 2024; https://zenodo.org/doi/10.5281/zenodo.4073010), and then an executable file that can be run has to be built, either on Windows or Linux. To build the executable file on Windows, 1) the LISFLOOD-FP folder should be opened in Visual Studio, 2) either the `x64-Debug` or `x64-Release` option should be selected from the dropdown menu near the toolbar at the top; and, 3) the `Build > Rebuild All` option should be clicked. If the `x64-Debug` option was selected, the executable file, named `lisflood.exe` and containing the GPU-MWDG2 solver, should be built and located in the folder at `LISFLOOD-FP\out\build\x64-Debug` or similar (or `LISFLOOD-FP\out\build\x64-Release` if the `x64-Release`

option was selected). To build on Linux, the steps are 1) navigating to the LISFLOOD-FP directory, 2) running `cmake -S . -B build` in the terminal; and, 3) running `cmake --build build`. The executable file, named `lisflood`, should be built and located in the `LISFLOOD-FP/build` directory.

After the executable file has been built, it can be run in order to run simulations of the "Monai valley" example using the GPU-MWDG2 solver. Before running the simulation, several input files must be prepared, which are listed in Table B.1. To prepare the input files, a number of Python scripts should be used that are available in the `monai` folder uploaded alongside the input files made publicly available online for reproducing the results and simulations reported in this paper (Chowdhury, 2024; https://doi.org/10.5281/zenodo.13909072). The usage of these Python scripts is as follows.

Table B.1: Input files needed to run simulations of the Monai valley example using the GPU-MWDG2 solver.

Input file	File name	Description
Digital elevation model	`monai.dem`	ASCII raster file containing the numerical values of the bathymetric elevation pixel-by-pixel.
Initial flow conditions	`monai.start`	ASCII raster file containing the numerical values of the initial water depth and discharge pixel-by-pixel.
Boundary conditions	`monai.bci`	Text file specifying where boundary conditions are enforced and what type (fixed versus time-varying).
Time series at boundaries	`monai.bdy`	Text file containing time series in case time-varying boundary conditions and/or point sources have been specified in the `.bci` file.
Stage locations	`monai.stage`	Text file containing the locations of virtual stage points where simulated time histories of the water depth are recorded.
Parameter file	`monai.par`	Text file containing parameters to access various solver and simulation features.

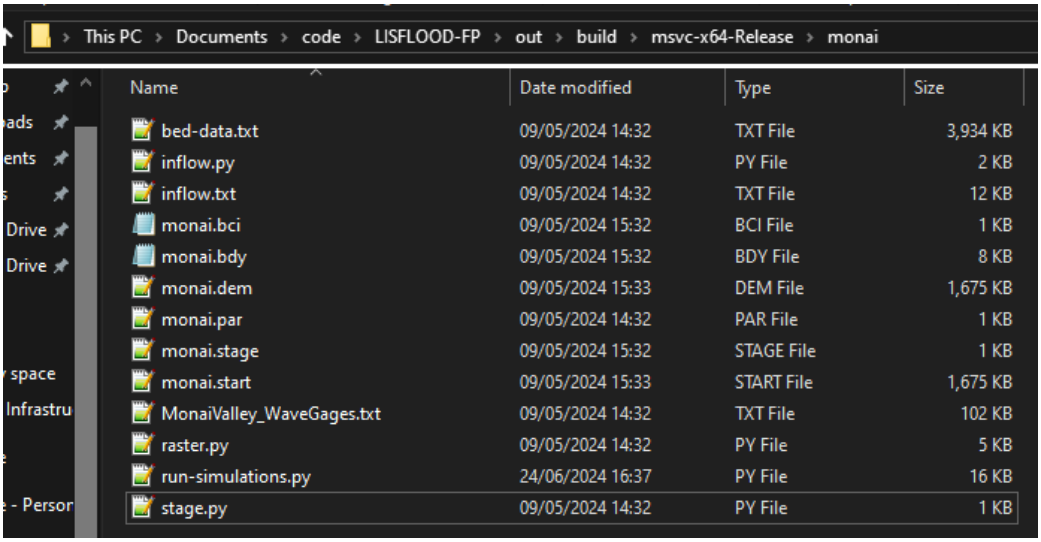

Figure B.1: Input files prepared for the "Monai valley" example after running the Python scripts available in the `monai` folder.

To prepare the input files for the Monai valley simulation using the Python scripts, 1) the `monai` folder should be copied to the same location as the `lisflood.exe` executable file, e.g. `LISFLOOD-FP\out\build\x64-Release` if on Windows or `LISFLOOD-FP/build` if on Linux, 2) the `monai` folder should be navigated to, 3) the `monai.stage` file should be generated by typing and running `python stage.py` in a command prompt, 4) the `monai.dem` and `monai.start` raster files should be generated by running `python raster.py`, 5) the `monai.bci` and `monai.bdy` files should be generated by running `python inflow.py`, 6) the parameter file `monai.par` should be prepared as shown in Figure 2; and, 7) the simulation should be run by typing and running `..\lisflood.exe monai.par` in a command prompt. Following steps (1) to (7) should result in the files shown in Figure B.1. Steps (1) to (7) can be performed as a fully automated process by running `python run-simulations.py`: this Python script will automatically prepare the input files (steps 3 to 6), run several simulations (step 7), and postprocess the results. Note that if `run-simulations.py` is run inside the downloaded `monai` folder before running any simulations, and with the `self.run()` function in the Python file commented out, it will reproduce the results in Sect. 3.1 (i.e. it will generate Figures 7 and 8 and the data for Table 3).